TWL Programming Manual

Version 1.4

CONFIDENTIAL

TWL-06-0017-001-D
Released: February 16, 2009

Confidential

Table of Contents

TWL-06-0017-001-D
Released: February 16, 2009

Figures

Tables

Revision History

Version	Date	Description
1.4	2009/01/29	• Corrected typos in Figures 3-1, 3-5, and 3-9.
1.3	2008/12/19	• Added a line in Figure 1-1 to connect the ARM7 bus and external memory interface. • Corrected error in DMA transfer speed from VRAM to main memory in Table 3-6. • Corrected cycle count in Figure 3-1. • Added cycles which include termination to Figures 3-3, 3-5, 3-7, 3-9, and 3-11. • Added Figures 3-4, 3-8 and 3-12 (Worst-case transfer sequence diagrams). • Added notes concerning the Display Selection Flags to the description of the Display Control Registers in Chapter 6. The notes describe the delay from the time a flag is set until display is enabled. • Added supplementary explanation of the excecution cycle count for geometry commands in Chapter 7. • Added information concerning the camera angle of view to Chapter 21. Changed “focal length” to “depth of field.” • Added a note to Chapter 23 concerning waits when reading.
1.2	2008/10/31	• Replaced one instance of the Nintendo racetrack logo and deleted several others. • Revised some addresses in Figure 1-2. • Revised the palette data in Appendix C and added a note.
1.1	2008/10/27	• Revised the Nintendo racetrack logo. • Revised the color data format in section 6.5, and added a note. • Revised the text in section 17.1.4, “Microphone Input Values.” • Revised the abbreviation for “command” from “COM” to “CMD in Chapter 22, “DSP.”
1.0	2008/10/01	• Initial version.

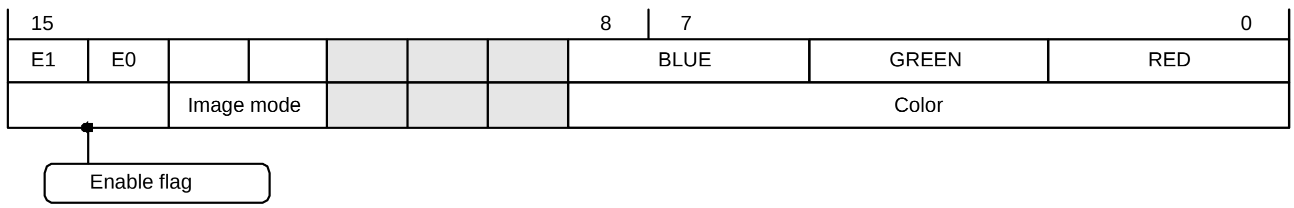
About the Notation Used in This Programming Manual

Registers

Detailed classifications are shown at the top of the register, while broader classifications are shown on the bottom. If text does not fit, then the description is shown below the register, as shown with the "enable flag" in the example below. Notation such as "d15" is used to refer to a specific bit (in this case the highest-order bit in a 16-bit register).

Example: TWL Register

Name: TWL Address: 0x04000??? Attribute: R/W Initial value: 0x0000



Bit lengths

Bit lengths for bytes, half-words, and words are defined as follows:

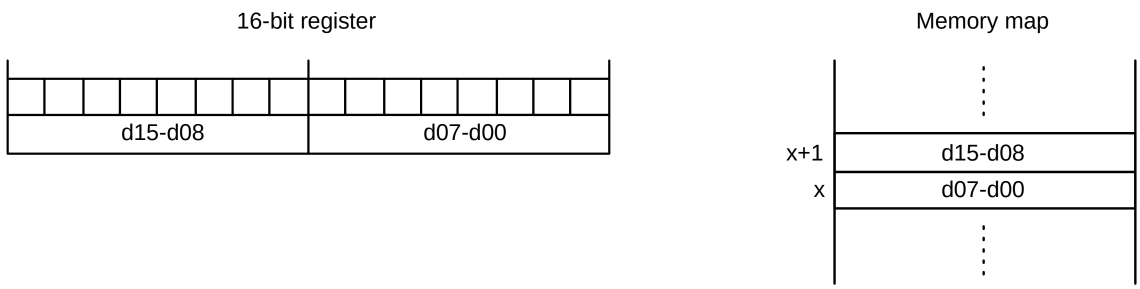
8-bit: Byte

16-bit: Half-word

32-bit: Word

Endian

TWL adopts the *little-endian* method. Therefore, in a 16-bit register, the address for d15–d08 is one more than the address for d07–d00.

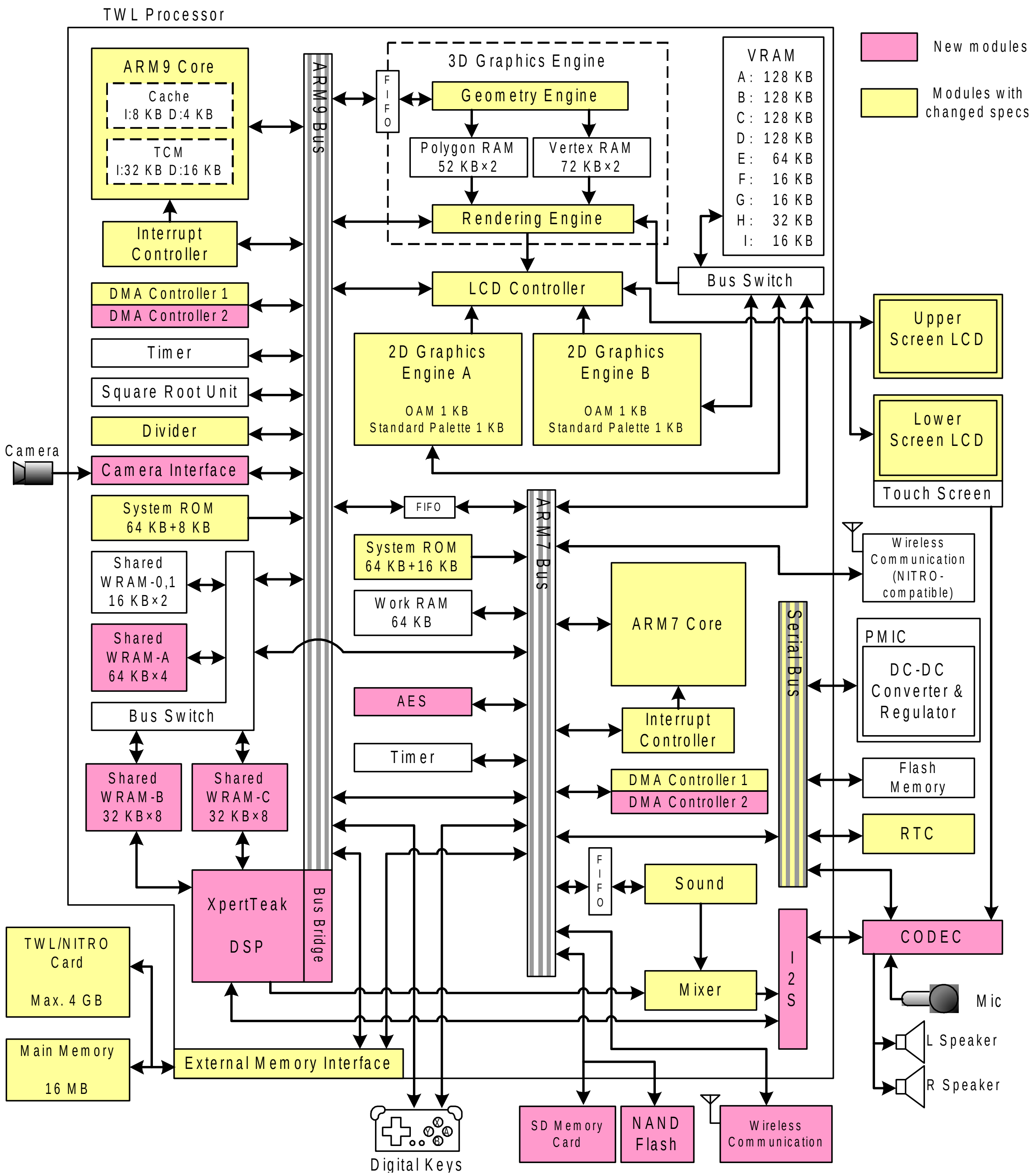


1 System

1.1 System Outline

The overall TWL system block diagram is shown in Figure 1-1.

Figure 1-1 : Overall System Block Diagram



1.1.1 TWL Processor

The TWL processor is a combined chip that includes both memory and TWL features such as ARM9 and ARM7 CPU cores, a DSP core, and 2D and 3D graphics engines.

The specifications of the TWL processor are as follows:

- The CMOS Multi CPU

Main processor core	ARM946E-S (134.056 MHz / 67.028 MHz)
Subprocessor core	ARM7TDMI (33.514 MHz)
DSP core	TeakDSP (134.056 MHz / 67.028 MHz / 33.514 MHz)

The operating frequency of the main processor core can be changed in the system configuration.

- Compatibility

Switches between TWL mode and NITRO compatibility mode.

(The AGB compatibility mode that was supported on NITRO is no longer available.)

- Graphics Engines

2D Graphics Engines A and B		33.514 MHz
3D Graphics Engine	Geometry Engine	33.514 MHz Maximum 4 million vertices per second 4 x 4 matrix computation 6-plane clipping Lighting (4 parallel light sources) Matrix stack Texture coordinate conversion Box culling test
	Rendering Engine	33.514 MHz Maximum 120 thousand polygons per second Maximum 30 million pixels per second Triangular and quadrilateral rendering Texture format 4-, 16-, and 256-color palette formats Bitmap format 4 x 4 texel compression format Translucent (A3I5, A5I3) format Texture size 8 x 8 to 1024 x1024 Alpha blending Alpha test Fog Toon shading Edge marking Anti-aliasing

Revisions to the circuits for each engine can be enabled or disabled using the system configuration.

- LCD Controller (built-in for two LCDs: the upper and lower screens)

Display Size	256 x 192 x RGB dots
Display Colors	262,144 colors (R:G:B = 6:6:6)
Dot Clock	5.586 MHz

- Memory

System ROM	ARM9 : For TWL : 64 KB (16K x 32 bit) For compatability : 8 KB (2K x 32 bit) ARM7 : For TWL : 64 KB (16K x 32 bit) For compatibility : 16 KB (4K x 32 bit)
TWL Processor Internal Work RAM	ARM9, ARM7 shared : 32 KB (8K x 32 bit) + 256 KB (16K x 32 bit x 4) ARM7 dedicated : 64 KB (16K x 32 bit) ARM9, ARM7, DSP instruction RAM shared : 256 KB (8K x 32 bit x 8) ARM9, ARM7, DSP data RAM shared : 256 KB (8K x 32 bit x 8)
VRAM	Total of 656 KB (128 KB + 128 KB + 128 KB + 128 KB + 64 KB + 16 KB + 16 KB + 32 KB + 16 KB) (Each VRAM can support 0 wait state access and byte write operations)
System Clock	33.514 MHz

- Sound

  ADPCM/PCM 16 channels (up to 6 channels for the PSG sound source and up to 2 channels for noise)

  The TWL includes sound capture capabilities (used for reverb, etc.).

  Mixing with DSP sound is possible.

- Timers

  ARM9 : 16-bit timer x 4

  ARM7 : 16-bit timer x 4

- DMA

  ARM9 : 4 channel x 2

  ARM7 : 4 channel x 2 + Sound DMA features

  There is a NITRO-compatible DMA controller, as well as a new DMA controller that can perform block transfers and allows you to choose the arbitration policies. Revisions to the circuits for the NITRO-compatible DMA controller can be enabled or disabled using the system configuration.

- Accelerator

  Divider (Revisions to the divider circuit can be enabled or disabled using the system configuration.)

  Square root unit

- External Memory Interface

  The TWL includes one slot for the TWL/DS Game Card interface (which adds card detection and hot swapping features) and one slot for the SD Card interface. (The Game Pak interface for AGB compatibility is no longer available.)

- NAND Flash Memory

  The TWL includes 256 MB of NAND Flash memory.

- Camera

  Supports capturing at 20 frames/second at VGA size or 30 frames/second at QVGA size.

  Includes a YUV to RGB conversion circuit (RGB555 format) and a trimming feature.

1.1.2 Main Memory

The main memory is 16 MB (expanded to 32 MB for TWL Debugger) and is connected to the TWL processor as an independent chip. When the system configuration are configured to NITRO compatibility mode, the memory is limited to 4 MB.

Because the TWL/NITRO card bus is not mapped to the CPU address space, applications and data must be executed after loading them into main memory.

ARM9 applications are transmitted from the TWL/NITRO card to main memory by system ROM at startup.

ARM7 applications are transmitted to the ARM7 dedicated work RAM at startup.

1.1.3 LCD

There are two LCD screens, an upper screen and a lower screen.

An overview of both LCD screen specifications is shown in Table 1-1.

Table 1-1 : Overview of LCD Screen Specifications

Features	Details
Display Resolution	256 x 192 pixels (Ratio 4:3)
Number of Displayable Colors	262,144 colors (RGB=6:6:6)
Screen Size	3.25 inches
Type	Transmissive
Backlight	Can be switched ON and OFF. Five steps of brightness can be adjusted via the IPL console configurations.

The main specifications are the same as the Nintendo DS Lite with the exception of the screen size and the number of brightness levels, but it is also possible to automatically control the screen to show either black or white when the power is turned off.

1.1.4 Digital Keys

The digital keys are START, SELECT, the + Keypad, A, B, X, Y, L, and R.

1.1.5 Touch Screen

The entire lower screen LCD is a resistive membrane touch panel that can obtain pixel-unit coordinates.

The TWL system comes equipped with a standard stylus.

1.1.6 Microphone

Either the Nintendo DS Series Headset or the built-in omnidirectional condenser microphone can be used.

Sound input from the microphone can be sampled.

1.1.7 RTC

The RTC keeps real time and handles timekeeping operations.

With the supported alarm feature, the RTC can wake up the TWL from Sleep Mode at a specified time.

1.1.8 Wireless Communications

The TWL includes two on-board wireless communications units capable of using the 2.4-GHz band. The following modes are available with the NITRO-compatible unit (NITRO wireless):

- Infrastructure Mode (Internet play) that allows connections to wireless LAN (IEEE 802.11b/g) access points
- Multi-Card Play that enables communications with up to 16 DS devices
- Single-Card Play that downloads games from a parent device to child devices that are not equipped with DS Game Cards

With the TWL wireless unit, only Infrastructure Mode (IEEE 802.11b/g) can be used, but the connections are faster and more secure than with the NITRO wireless unit).

1.1.9 TWL/NITRO Game Cards

A TWL/NITRO Game Card is a game card with TWL/NITRO-exclusive security features. A backup device can be included in addition to the ROM.

The TWL/NITRO Game Card is connected to the TWL processor via an external memory interface. For more information, refer to the Nintendo DS/TWL Game Card Manual.

1.1.10 Game Paks

AGB-compatibility mode, which was supported on NITRO, is no longer supported. When in NITRO compatibility mode, attempting to access a Game Pak will yield the same behavior as if no Game Pak were inserted.

1.1.11 SD Cards

The TWL includes one SD Card slot that can read SD (Secure Digital) Cards and SDHC (SD High Capacity) Cards with capacities between 2 and 32 GB. It does not support SPI mode, encrypted authentication (CPRM), or High-Speed mode.

1.1.12 NAND Flash Memory

TWL includes 256 MB of NAND Flash memory.

NAND Flash memory is used for saving NAND applications, saving the save data for NAND applications (including the system applications), and for saving photo data.

1.1.13 Camera

The camera can capture movies at 20 frames/second at VGA (640x480) size, or 30 frames/second at QVGA (320x240) size.

1.1.14 DSP (Digital Signal Processor)

The DSP accelerates image and sound processing. By using the work RAM, data processed by the DSP can be shared by the ARM9 and the ARM7. In addition, is also possible to mix the output from the sound circuit with sounds generated by the DSP and output the result.

1.2 Memory Map

The overall memory maps for the ARM9 and ARM7 on the production version of the TWL are shown in Figure 1-2. The yellow areas in the memory map indicate differences with the NITRO specifications, and the magenta areas indicate brand new portions that were added for TWL.

The access attributes for the ARM9 memory space are determined by the configuration of the protection units.

For more details, see "4.1 Protection Unit" on page 59.

- About the image

  A decoder converts the address output by the CPU to a memory address.

  Because the decoder does not normally decode all the address bits, when an address that is not mounted in memory is accessed, it is converted to the address of the memory located closest to the smallest part of the address. (See the note below.)

  Images are regions of the memory map that appear where nothing normally exists (regions differ from the physical memory).

  Even if the address is different on the CPU, the address is the same in memory. Therefore, accessing an image is the same as accessing the physical memory. Furthermore, when the image region is larger than the memory size, an image the same size as the physical memory is repeated.

  For example, the BG-VRAM physical memory for 2D Graphics Engine A occupies the 512 kilobytes in the 0x06000000 - 0x0607FFFF section in the Memory Map, but the image region occupies the 1536 KB of the 0x06080000 - 0x061FFFFF section, so there are three images.

  Note: There is some freedom in where the shared ARM9 and 7 internal work RAM can be physically positioned, as long as it is kept in the range 0x03000000-0x03FFFFFF.

  In addition, there are regions of undefined data that have no image (see Figure 1-2).

Note: Do not access the undefined data.

Figure 1-2 : ARM9 and ARM7 Overall Memory Maps

ARM9 Overall Memory Map

0xFFFF0000	System ROM (64 KB)
0x0E000000	Undefined Data
0x0D000000	Main Memory (16 MB when expanded)
0x0C000000	Main Memory Image (16 MB)
0x0A010000	Undefined Data
0x0A000000	Game Pak RAM (0-64 KB) (see note)
0x08000000	Game Pak ROM (32 MB) (see note)
0x07000800	OAM Image
0x07000400	2D Graphics Engine B OAM (1 KB)
0x07000000	2D Graphics Engine A OAM (1 KB)
0x068A4000	VRAM Image for LCDC
0x06800000	VRAM for LCDC (656 KB)
0x06620000	OBJ-VRAM Image
0x06600000	2D Graphics Engine B OBJ-VRAM (max.128 KB)
0x06440000	OBJ-VRAM Image
0x06400000	2D Graphics Engine A OBJ-VRAM (max. 256 KB)
0x06220000	BG-VRAM Image
0x06200000	2D Graphics Engine B BG-VRAM (max. 128 KB)
0x06080000	BG-VRAM Image
0x06000000	2D Graphics Engine A BG-VRAM (max. 512 KB)
0x05000800	Palette RAM Image
0x05000600	2D Graphics Engine B Palette RAM for OBJ (512 B)
0x05000400	2D Graphics Engine B Palette RAM for BG (512 B)
0x05000200	2D Graphics Engine A Palette RAM for OBJ (512 B)
0x05000000	2D Graphics Engine A Palette RAM for BG (512 B)
0x04000000	I/O Registers (see note)
0x03800000	ARM9, 7 Shared Internal Work RAM Image
0x03738000	ARM9, 7 Shared Internal Work RAM (max. 800 KB)
0x03000000	ARM9, 7 Shared Internal Work RAM Image
0x02FE0000	Data TCM (16 KB): Moveable (see note)
0x02000000	Main Memory (16 MB)
0x01FF8000	Instruction TCM (32 KB)
0x01000000	Instruction TCM Image
0x00000000	Undefined Data

ARM7 Overall Memory Map

0x0E000000	Undefined Data
0x0D000000	Main Memory (16 MB when expanded)
0x0C000000	Main Memory Image (16 MB)
0x0A010000	Undefined Data
0x0A000000	Game Pak RAM (0-64 KB) (see note)
0x08000000	Game Pak ROM (32 MB) (see note)
0x07000000	Undefined Data
0x06040000	Internal Expanded Work RAM Image
0x06000000	Internal Expanded Work RAM (max. 256 KB)
0x04810000	Undefined Data
0x04808000	Wireless Communications Wait State 1
0x04800000	Wireless Communications Wait State 0
0x04000000	I/O Registers (see note)
0x03810000	ARM7 Dedicated Internal Work RAM Image
0x03800000	ARM7 Dedicated Internal Work RAM (64 KB)
0x03738000	ARM7, 9 Shared Internal Work RAM (max. 800 KB)
0x03000000	ARM7, 9 Shared Internal Work RAM Image
0x02000000	Main Memory (16 MB)
0x00010000	Undefined Data
0x00000000	System ROM (64 KB)

Note: The I/O registers are different on ARM9 and ARM7.
Note: The Data TCM can be moved around, so the address in the figure above is just an example.
Note: The Game Pak region can be accessed, but write operations will be ignored, and reads will return fixed values.

1.3 Accessing Devices Connected to the Subprocessor

On TWL, as with NITRO, you must use the API to access devices connected to the subprocessor.

By using the API, you can access the device, regardless of what state the subprocessor is in.

The following devices connect to the subprocessor: wireless communications, a portion of the digital keys, the sound, Touch Screen, microphone, RTC, built-in flash memory, SD Card, NAND flash memory, and the AES encryption/decryption circuit.

- What is an API?

  An *Application Program Interface* (API) is a group of functions that increase efficiency when developing applications. In general, the API is used in low-level system calls and to control hardware.

Note: It is possible to access the registers related to the interface with the ARM7 subprocessor in ARM9, but these registers should not be accessed if using the API.

1.4 Startup Mode

Choose one of the following modes for the operation of your application. The mode is determined when the application is created and cannot be changed thereafter.

1.4.1 TWL Mode

In this mode, all TWL features can be used.

1.4.2 NITRO Compatibility Mode

In this mode, compatibility with NITRO is maintained.

Note: Because the AGB Game Pak slot has been eliminated, attempting to access a Game Pak will yield the same behavior as if no Game Pak were inserted. The method of sound output has also been changed.

1.5 Markets

As shown in Table 1-2, there are four TWL system configurations, depending on the market.

Table 1-2 : Differences in TWL Systems by Market

Market	Configurable Languages	Available Banner Font
Japan (JP)	Japanese	Hiragana, Katakana, Alphabetical, Signed Alphabetical, Numeric, etc.
North America (US)	English, French, Spanish	Same as above
Europe (EU)	English, French, Spanish, German, Italian	Same as above
Australia (AU)	English	Same as above

Note: With NITRO, the same systems were used for all regions (Japan, North America, Europe, and Australia), and the user could freely change which language was used. With TWL, there are four types of TWL systems (one for each region), and the languages that can be selected are different for each region as shown in Table 1-2.

2 Differences from NITRO

TWL is compatible with NITRO, but there are a number of differences between TWL and NITRO. For example, new modules have been added, the Game Pak slot has been eliminated, and it is possible to use circuits that correct hardware bugs. This chapter summarizes the differences between TWL and NITRO and explains the system configuration used to manage the modules used with TWL.

2.1 Newly Added Modules

The newly added modules are the camera, DSP, new shared WRAM, new DMA, new wireless LAN (TWL wireless), SD Card, NAND flash memory, and the AES encryption/decryption circuit. Among these, the new wireless LAN (TWL wireless), SD Card, NAND flash memory, and AES encryption/decryption circuit are connected to the subprocessor, so they can only be operated through the API.

2.2 Game Pak Slot

The Game Pak slot has been eliminated, but no changes have been made to the registers or the addresses it is mapped to. It is also possible to use these registers as the trigger to recover from sleep mode. However, writing is disabled for the addresses that the Game Pak slot is mapped to, and the data that is read from these registers will always be 0. Additionally, if the Game Pak IREQ/DREQ interrupt permission flag is set to 1, it will not be possible to transition to sleep mode.

2.3 Changed Modules

The analog-digital conversion that was previously performed with PMIC on NITRO is now handled by a module called “CODEC,” and the specifications for the mixer output and microphone input have changed (no changes have been made to the sound circuit itself). For details on the mixer, see "14.1.3 Mixer" on page 334. For details on the microphone, see "17 Microphone" on page 353.

2.4 System Configuration

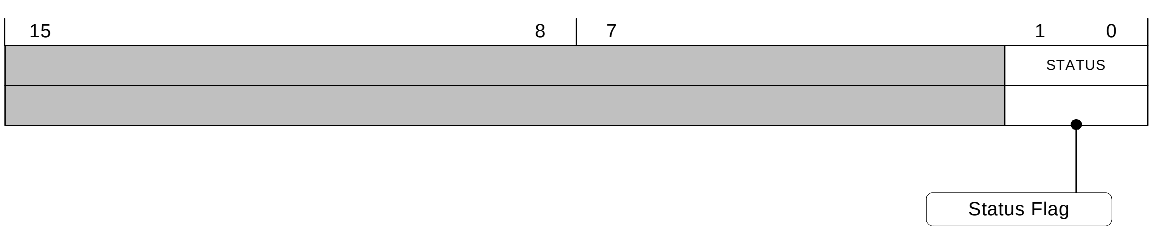
In TWL mode, it is possible to control both the provision of clock signals and access to the newly added modules by means of the system configuration registers (SCFG_*).

2.4.1 System ROM

The register below lets you look up the status of the system ROM used by the ARM9.

System Configuration: ROM Status Register

Name	Address	Attribute	Initial Value
SCFG_A9ROM	0x04004000	R	0x0000



- STATUS[d01–d00] : Status Flag

00	NITRO compatibility mode

01	TWL mode

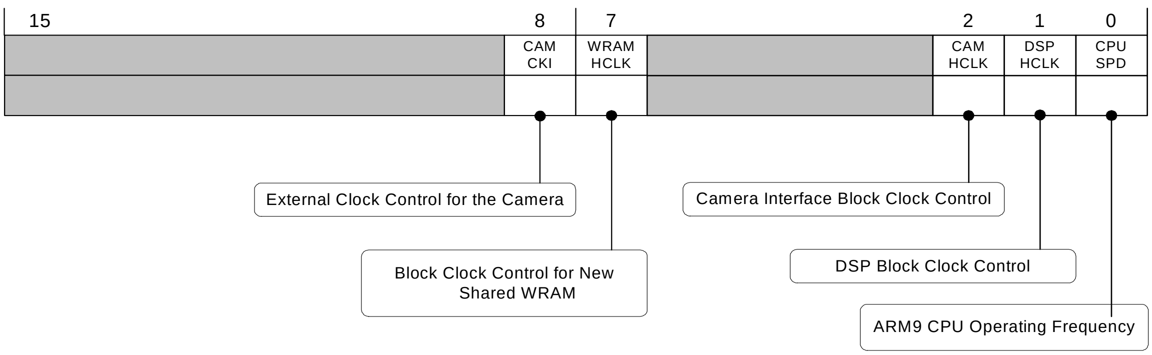
These bits are intended for reading the values configured in the subprocessor, so they cannot be written to.

2.4.2 New Block Clock Control

The register below controls the provision of a clock signal to the camera, new shared WRAM, and the DSP, and also controls the operating frequency of the ARM9.

System Configuration: <u>New Block Clock Control Register</u>

Name	Address	Attribute	Initial Value
SCFG_CLK	0x04004004	R/W	0x0084



- CAMCKI[d08] : External Clock Control for the Camera

0	Disable
1	Enable (Outputs at 16.76 MHz)

- WRAMHCLK[d07] : Block Clock Control for New Shared WRAM

0	Stop clock
1	Provide clock

These bits are intended for reading the values configured in the subprocessor, so they cannot be written to.

- CAMHCLK[d02] : Camera Interface Block Clock Control

0	Stop clock
1	Provide clock

- DSPHCLK[d01] : DSP Block Clock Control

0	Stop clock
1	Provide clock

- CPUSPD[d00] : ARM9 CPU Operating Frequency

0	Same as NITRO (67.03 MHz)

1	2x Speed Operation (134.06 MHz)

To be safe, only stop the clock after each of the modules has been stopped.

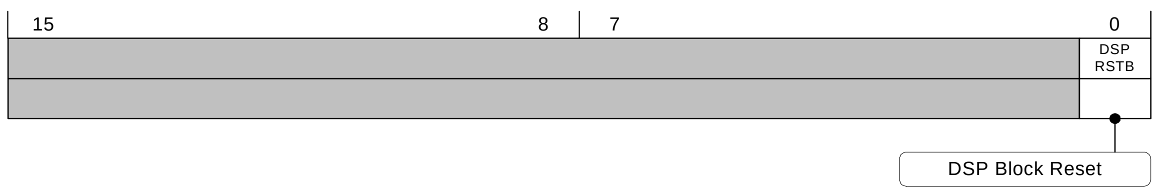
When changing the operating frequencing of the ARM9, overwrite the bit from the program in ITCM, and then let a minimum of 8 cycles go by in a dummy loop in ITCM. (Do not allow any access to the ARM9 bus during this time.)

2.4.3 New Block Reset

When applying a reset to the DSP block, manipulate the register shown below.

System Configuration: <u>New Block Clock Control Register</u>

Name	Address	Attribute	Initial Value
SCFG_RST	0x04004006	R/W	0x0000



- DSPRSTB[d00] : DSP Block Reset

0	Apply a reset to the DSP block
1	Undo the reset to the DSP block

2.4.4 Extended Features Control

When switching between enabling and disabling the new blocks, or when using the circuits that correct problems with NITRO, manipulate the register shown below.

System Configuration: Extended Features Control Register

Name	Address	Attribute	Initial Value
SCFG_EXT	0x04004008	R/W	0x8307F100

31		24	23	18	17	16
CFG		WRAM		DSP	CAM	DMAC
		New Block Access Control				

Enable/Disable System Control Block

15	14	13	12		8	7		4	3	2	1	0
PSRAM		VRAM	LCDC		INTC	MC		DIV	2DE	REN	GEO	DMA
NITRO Block Extended Features Control						NITRO Block Circuit Revisions Control						

- CFG[d31] : System Control Block Enable/Disable

0	Access disabled
1	Access enabled

Writing a 0 will prevent access to all registers in the system control block (SCFG_* and MBK1-9: 0x04004000-0x04004063) thereafter. Be sure to configure all system configuration-related registers appropriately before writing a 0.

- [d24-d16] : New Block Access Control

Even if access is disabled to each block, any values that were set in the registers prior to that point will be valid. To be safe, be sure to change these bits only when no access to each block is being generated.

- WRAM[d24] : Access to New Shared WRAM

0	Access disabled
1	Access enabled

- DSP[d18] : Access to DSP Block

0	Access disabled
1	Access enabled

- CAM[d17] : Access to Camera Interface

0	Access disabled
1	Access enabled

- DMAC[d16] : Access to New DMA Controller

0	Access disabled
1	Access enabled

- [d15-d08] : NITRO Block Extended Feature Control
  For details on the extended features, refer to "2.4.4.1 Extended Features" on page 14.
- PSRAM[d15-d14] : Main Memory Limitations

00 **01**	Limit to 4 MB boundary
10	Limit to 16 MB boundary
11	Limit to 32 MB boundary

- VRAM[d13] : Extended VRAM Access

0	Same as NITRO
1	Extended VRAM access

- LCDC[d12] : Extended LCDC Circuit

0	Same as NITRO
1	Extended LCDC circuit

- INTC[d08] : Extended Interrupt Circuit

0	Same as NITRO
1	Extended interupt circuit

- [d07-d00] : NITRO Block Circuit Revision Control
  For details on the revised features, refer to "2.4.4.2 Revised Features" on page 14.
- MC[d07] : TWL / NITRO Card Interface Circuit Revisions

0	Same as NITRO
1	Use the revised circuits

- DIV[d04] : Divider Circuit Revisions

0	Same as NITRO
1	Use the revised circuits

- 2DE[d03] : 2D Engine Circuit Revisions

0	Same as NITRO
1	Use the revised circuits

- REN[d02] : Renderer Circuit Revisions

0	Same as NITRO
1	Use the revised circuits

- GEO[d01] : Geometry Circuit Revisions

0	Same as NITRO
1	Use the revised circuits

- DMA[d00] : NITRO-Compatible DMA Circuit Revisions

0	Same as NITRO
1	Use the revised circuits

2.4.4.1 Extended Features

Blocks that support bits will have extended features as shown below. To be safe, only change the bits when the various blocks are not operating.

- Main Memory

  The size of the main memory can be limited to 4 MB (for NITRO compatibility) or extended to 16 MB or 32 MB boundaries.

Figure 2-1 Differences Due to Main Memory Size

	4 MB (NITRO-compatible)	16 MB	32 MB
0x0E000000			
0x0D000000	ALL 0	Main Memory Image (16 MB)	Extended Main Memory (16 MB)
0x0C000000	ALL 0	Main Memory Image (16 MB)	Main Memory Image (16 MB)
0x03000000			
	Main Memory Image (4 MB)	Main Memory (16 MB)	Main Memory (16 MB)
	Main Memory Image (4 MB)		
	Main Memory Image (4 MB)		
0x02000000	Main Memory (4 MB)		

- VRAM Access

  Will support a 32-bit bus width, and one-byte writes are possible.

- LCDC Circuit

  An LCD initialization signal will be added to the display status register (DISPSTAT).

- Interrupt Circuit

  Interrupt flags that support the new blocks will be added (New DMA 0-3, Camera, DSP). If this extended feature is turned off, the interrupt permission flags for the corresponding extended parts will all be initialized to 0 by the hardware. For details, refer to "10 Interrupts" on page 311.

2.4.4.2 Revised Features

When using the circuits that revise the defects, you must set the corresponding bits to 1 and switch to using an SDK that supports the circuit revisions. It should be noted that changing the SDK could cause subtle changes in an application's behavior, and routines that are coded with tight timing could be processed in a different order. To be safe, only change the bits when a given block is not operating.

- TWL / NITRO Card Interface

  Bugs that used to occur when writing to multiple pages or writing 32-bit indefinite data have been fixed.

- NITRO-Compatible DMA

  A bug in which high-priority DMA would run out of control if DMA was started on multiple channels in parallel on the ARM9 system bus has been fixed. Additionally, a bug in which it was impossible to specify an I/O register or the Game Pak bus as the source address for DMA0 has been fixed.

- Geometry Circuit
  - BoxTest

    A bug in which the box used in the test will be judged to be out of view if all six of its faces are outside the view volume has been fixed. Additionally, a bug that caused incorrect results to be returned when polygon reduction occurred has been fixed.
  - TexImageParam Command

    A bug in which attributes set with TexImageParam0/1 will be affected by the previous polygon has been fixed.
  - Command FIFO

    A bug that causes unintended geometry commands to be inserted by a particular data array when packing commands or continuously writing to the geometry FIFO using the CPU has been fixed.
  - 1-Pixel Polygons

    A bug that forces vertex sharing to be cancelled when 1-pixel polygons are removed from a group of linked polygons has been fixed.
  - Clipping

    A bug in which the values for the blue and green components of a vertex will turn to 0 (color distortion) when a vertex is clipped has been fixed.
- Renderer Circuit
  - 4x4 Compressed Textures

    A bug that caused 4x4 compressed textures with linear interpolation enabled to appear slightly darker than the corresponding texture with no interpolation (even for the exact same color) has been fixed.
  - Shadow Polygons

    A bug that caused shadow polygons for masks and shadow polygons for rendering to be rendered differently has been fixed.
- 2D Graphics Engine

  A bug that caused the window to extend vertically if the upper-left Y-coordinate was set to 6 or less with the BG window feature and a bug that made it impossible to specify a window that covers the entire display area in the X direction have been fixed.
- Divider Bug Revisions

  A bug that caused the Division by Zero Error flag not to function properly unless all 64 bits of the divisor (denominator) are zero has been fixed.

2.4.5 New Shared WRAM

Work RAM has been added, which can be shared by the ARM9, the subprocessor, and the DSP. Please refer to "3.2.2 Work RAM" on page 40 for more details.

3 Memory

Table 3-1 shows the configuration and specifications of the memory built into TWL.

Table 3-1 : Memory Configuration and Specifications

Memory Type	Bus Width	Access Cycle	Bit Width that Allows DMA Access		Bit Width that Allows Main Processor Access		Bit Width that Allows New DMA Access	
			Read	Write	Read	Write	Read	Write
OAM	32	1	16/32	16/32	8/16/32	16/32	32	32
VRAM	32 *1 /16	1	16/32	16/32	8/16/32	8 *1 / 16/32	32	32
Palette RAM	16	1	16/32	16/32	8/16/32	16/32	32	32
I/O Registers	32	1	16/32	16/32	8/16/32	8/16/32	32	32
Internal Work RAM	32	1	16/32	16/32	8/16/32	8/16/32	32	32
Main Memory	16	1st R:5 / W:4 2nd 1	16/32	16/32	8/16/32	8/16/32	32	32
System ROM	32	1	-	-	8/16/32	-	-	-
TCM/Cache	32	½ / ¼ *2	-	-	8/16/32	8/16/32	-	-

The values given for the number of access cycles correspond to a bus frequency of 33.514 MHz.

These values are for when memory is accessed in a bit width that is equal to or less than the bus width. When memory is accessed in a bit width that is larger than the bus width, the number of access cycles is limited to the bit width divided by the bus width.

Note 1: In TWL mode, the system configuration can be used to set the VRAM bus width to 32 bits and to enable 1-byte writes (configured using the SCFG_EXT register).

Note 2: In TWL mode, the system configuration can be used to set the operating frequency of the main processor to 2x that of NITRO (134.06 MHz) (configured using the SCFG_CLK register).

- **Transfer speeds between memories**

  DMA transfer speeds between memories can be calculated from the bus width and the access cycle values shown in Table 3-1. Table 3-2 shows an example of DMA transfers between Internal Work RAM and VRAM.

Table 3-2 : DMA Transfer Speeds between Internal Work RAM and VRAM (When the VRAM Bus Width Is 16 Bits)

Transfer Memory	DMA Transfer Bit Count	Cycles Used for Reading	Cycles Used for Writing	Total Number of Cycles	Transfer Speed (MB/sec)
From Internal Work RAM to VRAM	16	1	1	2	31.96
	32	1	2	3	42.62
From VRAM to Internal Work RAM	16	1	1	2	31.96
	32	2	1	3	42.62

Table 3-3 : DMA Transfer Speeds between Internal Work RAM and VRAM (When the VRAM Bus Width Is 32 Bits)

Transfer Memory	DMA Transfer Bit Count	Cycles Used for Reading	Cycles Used for Writing	Total Number of Cycles	Transfer Speed (MB/sec)
From Internal Work RAM to VRAM	16	1	1	2	31.96
	32	1	1	2	63.92
From VRAM to Internal Work RAM	16	1	1	2	31.96
	32	1	1	2	63.92

- Main memory transfer speeds

  Main memory can function as a look-ahead buffer by setting DMA for reading as shown in Table 3-4.

Table 3-4 : DMA Settings to Function as a Look-Ahead Buffer

Property to Set	Set Value
Transfer Bit Count	32 bits
How to Update Source Address	Increment
Destination Address	Not main memory

Capable of holding 16 bits, the look-ahead buffer reads from main memory while data is being written to the destination memory.

Table 3-5 and Table 3-6 show the DMA transfer speeds between main memory and internal Work RAM, and between main memory and VRAM. The asterisk (*) denotes a shortening of the total number of cycles due to the look-ahead buffer. For more details about the look-ahead buffer, see "3.1.1 Main Memory" on page 21.

Table 3-5 : DMA Transfer Speeds Between Main Memory and Internal Work RAM

Transfer Memory	DMA Transfer Bit Count	Cycles Used for Reading	Cycles Used for Writing	Total Number of Cycles	Transfer Speed (MB/sec)
From Main Memory to Internal Work RAM	16	1st 5 2nd- 1	1	1st 6 2nd- 2	2nd- 31.96
	32	1st 6 2nd- 2	1	1st 7 * 2nd- 2	2nd- 63.92
From Internal Work RAM to Main Memory	16	1	1st 4 2nd- 1	1st 5 2nd- 2	2nd- 31.96
	32	1	1st 5 2nd- 2	1st 6 2nd- 3	2nd- 42.62

Table 3-6 : DMA Transfer Speeds Between Main Memory and VRAM (When the VRAM Bus Width Is 16 Bits)

Transfer Memory	DMA Transfer Bit Count	Cycles Used for Reading	Cycles Used for Writing	Total Number of Cycles	Transfer Speed (MB/sec)
From Main Memory to VRAM	16	1st 5 2nd- 1	1	1st 6 2nd- 2	2nd- 31.96
	32	1st 6 2nd- 2	2	1st 8 * 2nd- 3	2nd- 42.62
From VRAM to Main Memory	16	1	1st 4 2nd- 1	1st 5 2nd- 2	2nd- 31.96
	32	2	1st 5 2nd- 2	1st 7 2nd- 4	2nd- 31.96

Table 3-7 : DMA Transfer Speeds Between Main Memory and VRAM (When the VRAM Bus Width Is 32 Bits)

Transfer Memory	DMA Transfer Bit Count	Cycles Used for Reading	Cycles Used for Writing	Total Number of Cycles	Transfer Speed (MB/sec)
From Main Memory to VRAM	16	1st 5 2nd- 1	1	1st 6 2nd- 2	2nd- 31.96
	32	1st 6 2nd- 2	1	1st 7 * 2nd- 2	2nd- 63.92
From VRAM to Main Memory	16	1	1st 4 2nd- 1	1st 5 2nd- 2	2nd- 31.96
	32	1	1st 5 2nd- 2	1st 6 2nd- 3	2nd- 42.62

Transfer speeds by the CPU tend to be slower than calculated because the actual transfer speed is related to the time it takes to execute commands and get to the bus.

3.1 External Memory

TWL/NITRO Game Card refers to the hardware for the TWL/NITRO Game Card slot.

EXMEMCNT: External Memory Control Register

Name: EXMEMCNT   Address: 0x04000204   Attribute: R/W   Initial Value: 0x0000

15	14			11			8	7							0
EP	IFM			MP											
Main Memory				TWL/NITRO Game Card											

- [d15, d14]: Main Memory: Settings related to main memory
  - EP[d15]: Select the CPU priority

    This defines which CPU has priority when ARM9 and ARM7 access main memory at the same time.

0	ARM9 priority
1	ARM7 priority

  - IFM[d14]: Interface mode switch flag

0	Asynchronous mode (this setting prohibited)
1	Synchronous mode

Note: You must set this to Synchronous mode.

- [d11]: Setting for the TWL/NITRO Game Card
  - MP[d11]: Select the CPU with access rights

0	ARM9
1	ARM7

3.1.1 Main Memory

3.1.1.1 Look-Ahead Buffer

A look-ahead buffer is implemented as the interface to main memory. This look-ahead buffer is shared by ARM9 DMA and ARM7 DMA.

When a 32-bit DMA transfer is conducted from a source address in main memory to a destination address other than main memory, the write time to the destination address is used to read the next data in 16-bit units. This method enables 32-bit reads to be conducted in a single read cycle.

Note that the look-ahead buffer cannot be used for other types of access, such as reading by the CPU or a 16-bit DMA transfer.

3.1.1.2 Burst Mode

In Burst mode, the sequential access of a half-word (16-bit width) is conducted in one cycle.

Table 3-8 shows the basic access cycles for random access and sequential access in Burst mode.

Table 3-8 : Basic Access Cycles

	Read	Write
Random	5 cycles	4 cycles
Sequential	1 cycle	1 cycle

3.1.1.2.1 Burst Access Conditions

In Burst mode, sequential access is called *burst access*.

Burst access is used during DMA transfer when either the source or the destination address is the main memory and the address update method for main memory is set to increment.

Because DMA transfers from main memory to main memory are all first accesses, routing through internal RAM provides fast transfer speeds.

In addition, burst access is also used when continuous transfers using LDM or STM instructions are executed.

3.1.1.2.2 Conditions for Inserting Waits

When a span of 236 half-words occurs, a wait of three cycles, called a *termination*, is inserted.

When the address is defined from 0 in units of 16 half-words, waits are also inserted if access starts from address 13, 14, or 15, as shown in Table 3-9 (a maximum of 3 waits are inserted). Processes are thus more efficient if data are located from a 4 half-word boundary (8-byte boundary).

Table 3-9 : Inserting Waits According to the Access Start Address

First Cycle	2	3	4	5	6	7	8	9	...
13	14	15	wait	16	17	18	19	20	...
14	15	wait	wait	16	17	18	19	20	...
15	wait	wait	wait	16	17	18	19	20	...

3.1.1.2.3 Main Memory DMA Transfer Cycles

3.1.1.2.3.1 32-Bit DMA Transfer Cycle from Main Memory to Work RAM

Because main memory has a 16-bit bus and Work RAM has a 32-bit bus, data read in half-word units is written in word units. While writing in word units, the look-ahead buffer reads more half-word data. Transfers to the geometry command FIFO are the same as transfers to Work RAM.

- Basic Cycles

  Figure 3-1 shows the basic cycles of the transfer sequence from main memory to Work RAM.

Figure 3-1 : Transfer Sequence from Main Memory to Work RAM (Basic Cycles)

Cycles	1	2	3	4	5	6	7	8	9	10	11	12	13
Work RAM							1W		2W		3W		4W
Main Memory	wait	wait	wait	wait	1LR	1HR	2LR	2HR	3LR	3HR	4LR	4HR	5LR

...	237	238	239	240	241	242	243	244	245	246	247	248	249
...			117W					118W		119W		120W	
...	117LR	117HR	T	T	T	118LR	118HR	119LR	119HR	120LR	120HR	121LR	121HR

- Worst Case

  Depending on the main memory address where access starts, a wait known as a termination may occur. In these cases, the access immediately after the termination becomes the first access.

  Figure 3-2 shows the worst-case transfer sequence from main memory to Work RAM.

Figure 3-2 : Transfer Sequence from Main Memory to Work RAM (Worst Case)

Cycles	1	2	3	4	5	6	7	8	9	10	11	12	13
Work RAM							1W						
Main Memory	wait	wait	wait	wait	1LR	1HR	T	T	T	wait	wait	wait	wait

14	15	16	17	18	19	20	21	22	23	24	25	26
		2W		3W		4W		5W		6W		7W
2LR	2HR	3LR	3HR	4LR	4HR	5LR	5HR	6LR	6HR	7LR	7HR	8LR

wait	Wait
T	Termination
nLR	Reads the lower 16 bits of the *n*th 32-bit data
nHR	Reads the upper 16 bits of the *n*th 32-bit data
nW	Writes the *n*th 32-bit data

3.1.1.2.3.2 Cycles for 32-bit DMA Transfers from Work RAM to Main Memory

- Basic Cycles

  Figure 3-3 shows the basic cycles of the transfer sequence from Work RAM to main memory.

Figure 3-3 : Transfer Sequence from Work RAM to Main Memory (Basic Cycles)

Cycles	1	2	3	4	5	6	7	8	9	10	11	12	13
Work RAM	1R						2R			3R			4R
Main Memory		wait	wait	wait	1LW	1HW		2LW	2HW		3LW	3HW	

...	352	353	354	355	356	357	358	359	360	361
...	117R									118R
...	wait	T	T	T	wait	wait	wait	117LW	117HW	

- Worst Case

  Depending on the main memory address where access starts, a wait known as a termination may occur. In these cases, the access immediately after the termination becomes the first access.

  Figure 3-4 shows the worst case transfer sequence from Work RAM to main memory.

Figure 3-4 : Transfer Sequence from Work RAM to Main Memory (Worst Case)

Cycles	1	2	3	4	5	6	7	8	9	10	11	12	13
Work RAM	1R						2R						
Main Memory		wait	wait	wait	1LW	1HW	wait	T	T	T	wait	wait	wait

14	15	16	17	18	19	20	21
		3R			4R		
2LW	2HW		3LW	3HW		4LW	4HW

wait	Wait
T	Termination
nR	Reads the *n*th 32-bit data
nLW	Writes the lower 16 bits of the *n*th 32-bit data
nHW	Writes the upper 16 bits of the *n*th 32-bit data

3.1.1.2.3.3 Cycles for 32-bit DMA Transfers from Main Memory to VRAM (16-Bit Width)

If the bus width for VRAM is set to 16 bits in the system configuration, data read in half-word units is written in half-word units because main memory and VRAM both have a 16-bit-width bus. While writing in half-word units, the look-ahead buffer reads more data from main memory.

- Basic Cycle

  Figure 3-5 shows the basic cycles of the transfer sequence from main memory to VRAM.

Figure 3-5 : Transfer Sequence from Main Memory to VRAM (Basic Cycles, 16-Bit Width)

Cycles	1	2	3	4	5	6	7	8	9	10	11	12	13
VRAM							1LW	1HW		2LW	2HW		3LW
Main memory	wait	wait	wait	wait	1LR	1HR	2LR		2HR	3LR		3HR	4LR

...	352	353	354	355	356	357	358	359	360	361	362
...			117LW	117HW							
...	117LR	117HR	T	T	T	wait	wait	wait	wait	118LR	118HR

- Worst Case

  Depending on the main memory address where access starts, a wait known as a termination may occur. In these cases, the access immediately after the termination becomes the first access.

  Figure 3-6 shows the worst-case transfer sequence from main memory to VRAM.

Figure 3-6 : Transfer Sequence from Main Memory to VRAM (Worst Case, 16-Bit Width)

Cycles	1	2	3	4	5	6	7	8	9	10	11	12	13
VRAM							1LW	1HW					
Main Memory	wait	wait	wait	wait	1LR	1HR	T	T	T	wait	wait	wait	wait

14	15	16	17	18	19	20	21	22	23	24	25	26
		2LW	2HW		3LW	3HW		4LW	4HW		5LW	5HW
2LR	2HR	3LR		3HR	4LR		4HR	5LR		5HR	6LR	

wait	Wait
T	Termination
nLR	Reads the lower 16 bits of the *n*th 32-bit data
nHR	Reads the upper 16 bits of the *n*th 32-bit data
nLW	Writes the lower 16 bits of the *n*th 32-bit data
nHW	Writes the upper 16 bits of the *n*th 32-bit data

3.1.1.2.3.4 Cycles for 32-bit DMA Transfers from VRAM (16-Bit Width) to Main Memory

- Basic Cycles

  Figure 3-7 shows the basic cycles of the transfer sequence from VRAM to main memory.

Figure 3-7 : Transfer Sequence from VRAM to Main Memory (Basic Cycles, 16-Bit Width)

Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13
VRAM	1LR	1HR						2LR	2HR			3LR	3HR
Main Memory			wait	wait	wait	1LW	1HW			2LW	2HW		

...	468	469	470	471	472	473	474	475	476	477	478	479	480
...	117LR	117HR			118LR	118HR							
...			117LW	117HW	wait	T	T	T	wait	wait	wait	118LW	118HW

- Worst Case

  Depending on the main memory address where access starts, a wait known as a termination may occur. In these cases, the access immediately after the termination becomes the first access.

  Figure 3-8 shows the worst case 32-bit transfer sequence from VRAM to main memory.

Figure 3-8 : 32-Bit Transfer Sequence from VRAM (16-Bit Width) to Main Memory (Worst Case)

Cycle	1	2	3	4	5	6	7	8	9	10	11	
VRAM	1LR	1HR						2LR	2HR			...
Main Memory			wait	wait	wait	1LW	1HW	wait	T	T	T	...

12	13	14	15	16	17	18	19	20
					3LR	3HR		
wait	wait	wait	2LW	2HW			3LW	3HW

wait	Wait
T	Termination
nLR	Reads the lower 16 bits of the *n*th 32-bit data
nHR	Reads the upper 16 bits of the *n*th 32-bit data
nLW	Writes the lower 16 bits of the *n*th 32-bit data
nHW	Writes the upper 16 bits of the *n*th 32-bit data

3.1.1.2.3.5 32-Bit DMA Transfer Cycle from Main Memory to VRAM (32-Bit Width)

If the bus width for VRAM is set to 32 bits in the system configuration, data read in half-word units is written in word units because main memory has a 16-bit-width bus and VRAM has a 32-bit-width bus. While writing in word units, the look-ahead buffer reads next data from main memory.

This transfer cycle is the same as a transfer sequence from main memory to Work RAM.

- Basic Cycles

  Figure 3-9 shows the basic cycles of the transfer sequence from main memory to VRAM.

Figure 3-9 : Transfer Sequence from Main Memory to VRAM (Basic Cycles, 32-Bit Width)

Cycles	1	2	3	4	5	6	7	8	9	10	11	12	13
VRAM							1W		2W		3W		4W
Main Memory	wait	wait	wait	wait	1LR	1HR	2LR	2HR	3LR	3HR	4LR	4HR	5LR

...	237	238	239	240	241	242	243	244	245	246	247
...			117W								
...	117LR	117HR	T	T	T	wait	wait	wait	wait	118LR	118HR

- Worst Case

  Depending on the main memory address where access starts, a wait known as a termination may occur. In these cases, the access immediately after the termination becomes the first access.

  Figure 3-10 shows the worst-case transfer sequence from main memory to VRAM.

Figure 3-10 : Transfer Sequence from Main Memory to VRAM (Worst Case, 32-Bit Width)

Cycles	1	2	3	4	5	6	7	8	9	10	11	12	13
VRAM							1W						
Main Memory	wait	wait	wait	wait	1LR	1HR	T	T	T	wait	wait	wait	wait

14	15	16	17	18	19	20	21	22	23	24	25	26
		2W		3W		4W		5W		6W		7W
2LR	2HR	3LR	3HR	4LR	4HR	5LR	5HR	6LR	6HR	7LR	7HR	8LR

wait	Wait
T	Termination
nLR	Reads the lower 16 bits of the *n*th 32-bit data
nHR	Reads the upper 16 bits of the *n*th 32-bit data
nW	Writes the *n*th 32-bit data

3.1.1.2.3.6 Cycles for 32-bit DMA Transfers from VRAM (32-Bit Width) to Main Memory

Figure 3-11 shows the basic cycles of the transfer sequence from VRAM to main memory.

Figure 3-11 : Transfer Sequence from VRAM to Main Memory (Basic Cycles, 32-Bit Width)

Cycles	1	2	3	4	5	6	7	8	9	10	11	12	13
VRAM	1R						2R			3R			4R
Main Memory		wait	wait	wait	1LW	1HW		2LW	2HW		3LW	3HW	

...	352	353	354	355	356	357	358	359	360	361	362	363
...	117R			118R								
...		117LW	117HW	wait	T	T	T	wait	wait	wait	118LW	118HW

- Worst Case

  Depending on the main memory address where access starts, a wait known as a termination may occur. In these cases, the access immediately after the termination becomes the first access.

  Figure 3-12 shows the worst-case 32-bit transfer sequence from VRAM to main memory.

Figure 3-12 : 32-Bit Transfer Sequence from VRAM (32-Bit Width) to Main Memory (Worst Case)

Cycles	1	2	3	4	5	6	7	8	9	10	...
VRAM	1R						2R				...
Main Memory		wait	wait	wait	1LW	1HW	wait	T	T	T	...

12	13	14	15	16	17	18	19
					3R		
wait	wait	wait	2LW	2HW		3LW	3HW

wait	Wait
T	Termination
nLR	Reads the lower 16 bits of the *n*th 32-bit data
nHR	Reads the upper 16 bits of the *n*th 32-bit data
nLW	Writes the lower 16 bits of the *n*th 32-bit data
nHW	Writes the upper 16 bits of the *n*th 32-bit data

3.2 The TWL Processor's Internal Memory

3.2.1 VRAM

VRAM (A to I) does not have a fixed use, so it can be assigned for each application in the ways that make the most efficient use of memory resources. This ability is called *VRAM bank control*. Do not switch banks during access to VRAM.

Table 3-10 shows the options for VRAM use.

Table 3-10 : Options for VRAM Use

Use	VRAM / KB	A	B	C	D	E	F	G	H	I
	KB	128	128	128	128	64	16	16	32	16
LCDC		X	X	X	X	X	X	X	X	X
ARM7				X	X					
2D Graphics Engine A	BG-VRAM	X	X	X	X	X	X	X		
	OBJ-VRAM	X	X			X	X	X		
	BG Extended Palette Slot					X	X	X		
	OBJ Extended Palette Slot						X	X		
2D Graphics Engine B	BG-VRAM			X					X	X
	OBJ-VRAM				X					X
	BG Extended Palette Slot								X	
	OBJ Extended Palette Slot									X
3D Graphics (Rendering Engine)	Texture Image Slot	X	X	X	X					
	Texture Palette Slot					X	X	X		

(BG is screen data or character data. OBJ is character data.)

Memory assigned to LCDC, ARM7, BG-VRAM, and OBJ-VRAM is also mapped to the ARM9 bus, enabling memory to be read and written by ARM9. Memory assigned to the extended palette and texture slots is not mapped to the ARM9 bus.

Note 1: LCDC: The LCD controller (LCDC) handles this region. VRAM A to D can be used as memory for holding bitmap data during VRAM display mode and it can also be set as memory for writing bit-map data during captures. (For details, see "5.4.4.2 VRAM Display Mode" on page 87 and "5.5 Display Capture" on page 91.)

VRAM assigned to LCDC is uniquely mapped to the ARM9 bus. (When all A-to-I of VRAM is assigned to LCDC, it is mapped in a contiguous region of the ARM9 bus.) Because there is no access from the CPU when VRAM is allocated to the Extended Palette slot or the Texture Image/ Palette slot, you need to temporarily set data in LCDC to write data.

Note 2: BG-VRAM: This region stores BG screen data, character data, and bitmap data. Up to 512 KB for 2D Graphics Engine A or up to 128 KB for 2D Graphics Engine B can be assigned for this purpose.

Note 3: OBJ-VRAM: This region stores OBJ character data and bitmap data. Up to 256 KB for 2D Graphics Engine A or up to 128 KB for 2D Graphics Engine B can be assigned for this purpose.

Note 4: Extended palette slots: This memory space is the property of the 2D graphics engine, which references color data when BG and OBJ are displayed. The slots are not mapped in the CPU's memory space.

Note 5: Texture image slot and texture palette slot: This memory space is the property of the rendering engine inside the 3D graphics engine. The rendering engine references texel colors when textures are blended. The slots are not mapped in CPU memory space.

VRAMCNT: RAM Bank Control Register 0

Name: VRAMCNT Address: 0x04000240 Attribute: W Initial Value: 0x00000000

<table>
<tr><td>31</td><td></td><td></td><td>28 27</td><td>26 24</td><td>23</td><td></td><td></td><td>20 19</td><td>18 16</td><td>15</td><td></td><td></td><td>12 11</td><td></td><td>9 8</td><td>7</td><td></td><td></td><td>4 3</td><td></td><td>1 0</td></tr>
<tr><td>E</td><td></td><td></td><td>OFS</td><td>MST</td><td>E</td><td></td><td></td><td>OFS</td><td>MST</td><td>E</td><td></td><td></td><td>OFS</td><td></td><td>MST</td><td>E</td><td></td><td></td><td>OFS</td><td></td><td>MST</td></tr>
<tr><td colspan="5">VRAM-D</td><td colspan="5">VRAM-C</td><td colspan="6">VRAM-B</td><td colspan="6">VRAM-A</td></tr>
</table>

- VRAM-D, VRAM-C
  - E[d31][d23] Enable flag

0	Disable
1	Enable

- OFS[d28–d27][d20–d19]: Allocated addresses (Allocated to the addresses shown below according to the MST: Allocation options)

1. When MST = 000

   Allocated to LCDC and also mapped to ARM9 memory space

VRAM-C	0x06840000-0x0685FFFF
VRAM-D	0x06860000-0x0687FFFF

2. When MST = 001

   Allocated to 2D Graphics Engine A's BG and also mapped to ARM9 memory space

00	0x06000000-0x0601FFFF
01	0x06020000-0x0603FFFF
10	0x06040000-0x0605FFFF
11	0x06060000-0x0607FFFF

3. When MST = 010

   Mapped to ARM7 memory space, so not mapped to ARM9 memory space

00	0x06000000-0x0601FFFF
01	0x06020000-0x0603FFFF
10	Setting prohibited
11	Setting prohibited

4. When MST = 011

   Allocated to texture image slot, but not mapped to ARM9 memory space

   (See the texture image slot memory map in Figure 3-13.)

00	Texture image slot 0
01	Texture image slot 1
10	Texture image slot 2 (clear color image)
11	Texture image slot 3 (clear depth image)

5. When MST = 100

   Mapped to the following ARM9 memory spaces, no matter what the setting

VRAM-C	0x06200000-0x0621FFFF
VRAM-D	0x06600000-0x0661FFFF

- MST[d26–d24][d18–d16]: Allocation options

000	Allocate to LCDC
001	Allocate to 2D Graphic Engine A's BG
010	Allocate to ARM7
011	Allocate to 3D rendering engine's texture image
100	For VRAM-C: Allocate to 2D Graphic Engine B's BG For VRAM-D: Allocate to 2D Graphic Engine B's OBJ
101-111	Setting prohibited

Note: Although VRAM-C and VRAM-D can be allocated to the ARM7 subprocessor, to ensure that the subprocessor API operates correctly, do not change the register settings.

- VRAM-B, VRAM-A
  - E[d15][d07]: Enable flag

0	Disable
1	Enable

  - OFS[d12–d11][d04–d03]: Allocated addresses (Allocated to the addresses shown below according to the MST: Allocation options)

1. When MST = 00

   Allocated to LCDC and also mapped to ARM9 memory space

VRAM-A	0x06800000-0x0681FFFF
VRAM-B	0x06820000-0x0683FFFF

2. When MST = 01

   Allocated to the 2D Graphic Engine A 's BG and also mapped to the ARM9 memory space

00	0x06000000-0x0601FFFF
01	0x06020000-0x0603FFFF
10	0x06040000-0x0605FFFF
11	0x06060000-0x0607FFFF

3. When MST = 10

   Allocated to the 2D Graphic Engine A's OBJ and also mapped to the ARM9 memory space

00	0x06400000-0x0641FFFF
01	0x06420000-0x0643FFFF
10	Setting prohibited
11	Setting prohibited

4. When MST = 11

   Allocated to the texture image slot, but not mapped to ARM9 memory space

   (See the texture image slot memory map in Figure 3-13.)

00	Texture image slot 0
01	Texture image slot 1
10	Texture image slot 2 (clear color image)
11	Texture image slot 3 (clear depth image)

- MST[d09–d08][d01–d00]: Allocation options

00	Allocated to the LCDC
01	Allocated to the 2D Graphic Engine A's BG
10	Allocated to the 2D Graphic Engine A's OBJ
11	Allocated to the 3D rendering engine's texture image

Operation is not guaranteed when multiple VRAM blocks are mapped to the same address or assigned to the same slot.

Texture image slots are memory-mapped in the rendering engine as in Figure 3-13.

Figure 3-13 : Texture Image Slot Memory Map

0x00080000

0x00060000	Slot 3 (clear depth image)
0x00040000	Slot 2 (clear color image)
0x00020000	Slot 1
0x00000000	Slot 0

Texture image Slot 2 and Slot 3 also works as a clear image buffer to initialize the rendering buffer. (Read about initializing with a clear image in "7.3.3 Initializing the Rendering Buffers" on page 256).

WVRAMCNT: RAM Bank Control Register 1

Name: WVRAMCNT   Address: 0x04000244   Attribute: W   Initial value: 0x00000000

31						25	24	23			20 19	18 16	15			12 11	10 8	7					2 0
							BANK	E			OFS	MST	E			OFS	MST	E					MST
WRAM								VRAM-G					VRAM-F					VRAM-E					

- VRAM-G, VRAM-F
  - E[d23][d15]: Enable flag

0	Disable
1	Enable

- OFS[d20–d19][d12–d11]: Allocated addresses (Allocated to the addresses shown below according to the MST: Allocation options)

1. When MST = 000

   Allocated to LCDC and also mapped to ARM9 memory space

VRAM-F	0x06890000-0x06893FFF
VRAM-G	0x06894000-0x06897FFF

2. When MST = 001

   Allocated to the 2D Graphic Engine A's BG and also mapped to ARM9 memory space

00	0x06000000-0x06003FFF
01	0x06004000-0x06007FFF
10	0x06010000-0x06013FFF
11	0x06014000-0x06017FFF

3. When MST = 010

   Allocated to the 2D Graphic Engine A's OBJ and also mapped to ARM9 memory space

00	0x06400000-0x06403FFF
01	0x06404000-0x06407FFF
10	0x06410000-0x06413FFF
11	0x06414000-0x06417FFF

4. When MST = 011

   Allocated to the texture palette slot, but not mapped to ARM9 memory space.

   (See the texture palette slot memory map in Figure 3-14.)

00	Texture palette slot 0
01	Texture palette slot 1
10	Texture palette slot 4
11	Texture palette slot 5

5. When MST = 100

   Allocated to the 2D Graphic Engine A's BG extended palette slot, but not mapped to ARM9 memory space.

   (See the BG extended palette slot memory map in Figure 3-15.)

00	2D Graphic Engine A BG extended palette slots 0-1
01	2D Graphic Engine A BG extended palette slots 2-3
10	Setting prohibited
11	Setting prohibited

6. When MST = 101

   Allocated to the 2D Graphic Engine A's OBJ extended palette slot, but not mapped to ARM9 memory space.

   The lower 8 KB are allocated to the slot, but the upper 8 KB are invalidated.

   (See the OBJ extended palette slot memory map in Figure 3-16.)

- MST[d18–d16][d10–d08]: Allocation options

000	Allocated to the LCDC
001	Allocated to the 2D Graphic Engine A's BG
010	Allocated to the 2D Graphic Engine A's OBJ
011	Allocated to the 3D Rendering Engine's texture palette
100	Allocated to the 2D Graphic Engine A's BG extended palette
101	Allocated to the 2D Graphic Engine A's OBJ extended palette
110, 111	Setting prohibited

- VRAM-E
  - E[d07]: Enable flag

0	Disable
1	Enable

  - MST[d02–d00]: Allocation options

000	Allocated to the LCDC
001	Allocated to the 2D Graphic Engine A's BG
010	Allocated to the 2D Graphic Engine A's OBJ
011	Allocated to the 3D Rendering Engine's texture palette
100	Allocated to the 2D Graphic Engine A's BG extended palette
101-111	Setting prohibited

Note: VRAM-E mapping is fixed according to the MST settings shown below. (The offset cannot be changed.)

000	ARM9 addresses 0x06880000-0x0688FFFF
001	ARM9 addresses 0x06000000-0x0600FFFF
010	ARM9 addresses 0x06400000-0x0640FFFF
011	Texture palette slots 0-3
100	BG extended palette slots 0-3 (only the lower 32 KB)

The texture palette slots are mapped in the Rendering Engine (see Figure 3-14) when VRAM blocks E, F, and G are assigned. The BG palette slots are mapped as shown in Figure 3-15.

Figure 3-14 : Texture Palette Slot Memory Map

0x00018000	
0x00014000	Slot 5
0x00010000	Slot 4
0x0000C000	Slot 3
0x00008000	Slot 2
0x00004000	Slot 1
0x00000000	Slot 0

Figure 3-15 : BG Extended Palette Slot Memory Map

0x00008000	
0x00006000	Slot 3
0x00004000	Slot 2
0x00002000	Slot 1
0x00000000	Slot 0

Figure 3-16 shows the memory map when VRAM-F and -G are allocated to the OBJ extended palette.

Figure 3-16 : OBJ Extended Palette Slot Memory Map

0x00004000	
0x00002000	Slot 1
0x00000000	Slot 0

Proper operation is not guaranteed when multiple VRAM blocks are mapped to the same address or assigned to the same slot.

VRAM_HI_CNT: RAM Bank Control Register 2

Name: VRAM_HI_CNT Address: 0x04000248 Attribute: W Initial value: 0x0000

15					10	9	8	7					2	1	0
E						MST		E						MST	
VRAM-I								VRAM-H							

- VRAM-I, VRAM-H
  - E[d15][d07]: Enable flag

0	Disable
1	Enable

- MST[d09–d08][d01–d00]: Allocation options
  - For VRAM-H

00	Allocated to LCDC Mapped to main processor addresses 0x06898000-0x0689FFFF
01	Allocated to 2D Graphics Engine B's BG Mapped to main processor addresses 0x06200000-0x06207FFF
10	Allocated to 2D Graphics Engine B's BG extended palette Mapped to slots 0-3
11	Setting prohibited

  - For VRAM-I

00	Allocated to LCDC Mapped to main processor addresses 0x068A0000-0x068A3FFFF
01	Allocated to 2D Graphics Engine B's BG Mapped to main processor addresses 0x06208000-0x0620BFFF
10	Allocated to 2D Graphics Engine B's OBJ Mapped to main processor addresses 0x06600000-0x06603FFF
11	Allocated to 2D Graphics Engine B's OBJ extended palette Mapped to slots 0-1

Table 3-11 through Table 3-16 show the addresses allocated to the various VRAM blocks for the given MST and OFS bits.

Table 3-11 : VRAM-A and VRAM-B Allocations

<table>
<tr><th>MST</th><th>OFS
Allocation</th><th>00</th><th>01</th><th>10</th><th>11</th></tr>
<tr><td>00</td><td>ARM9</td><td colspan="4">VRAM-A: 0x06800000-0x0681FFFF
VRAM-B: 0x06820000-0x0683FFFF</td></tr>
<tr><td>01</td><td>2D Graphics Engine A</td><td>BG-VRAM (0x06000000-0x0601FFFF)</td><td>BG-VRAM (0x06020000-0x0603FFFF)</td><td>BG-VRAM (0x06040000-0x0605FFFF)</td><td>BG-VRAM (0x06060000-0x0607FFFF)</td></tr>
<tr><td>10</td><td>2D Graphics Engine A</td><td>OBJ-VRAM (0x06400000-0x0641FFFF)</td><td>OBJ -VRAM (0x06420000-0x0643FFFF)</td><td colspan="2">Setting prohibited</td></tr>
<tr><td rowspan="2">11</td><td rowspan="2">Texture Image</td><td rowspan="2">Slot 0</td><td rowspan="2">Slot 1</td><td>Slot 2</td><td>Slot 3</td></tr>
<tr><td colspan="2">(clear image)</td></tr>
</table>

Table 3-12 : VRAM-C and VRAM-D Allocations

<table>
<tr><th>MST</th><th>OFS
Allocation</th><th>00</th><th>01</th><th>10</th><th>11</th></tr>
<tr><td>000</td><td>ARM9</td><td colspan="4">VRAM-C: 0x06840000-0x0685FFFF
VRAM-D: 0x06860000-0x0687FFFF</td></tr>
<tr><td>001</td><td>2D Graphics Engine A</td><td>BG-VRAM (0x06000000-0x0601FFFF)</td><td>BG-VRAM (0x06020000-0x0603FFFF)</td><td>BG-VRAM (0x06040000-0x0605FFFF)</td><td>BG-VRAM (0x06060000-0x0607FFFF)</td></tr>
<tr><td>010</td><td>ARM7</td><td>ARM7 0x06000000-0x0601FFFF</td><td>ARM7 0x06020000-0x0603FFFF</td><td colspan="2">Setting prohibited</td></tr>
<tr><td rowspan="2">011</td><td rowspan="2">Texture Image</td><td rowspan="2">Slot 0</td><td rowspan="2">Slot 1</td><td>Slot 2</td><td>Slot 3</td></tr>
<tr><td colspan="2">(clear image)</td></tr>
<tr><td>100</td><td>2D Graphics Engine B</td><td colspan="4">VRAM-C: BG-VRAM (0x06200000-0x0621FFFF)
VRAM-D: OBJ-VRAM (0x06600000-0x0661FFFF)</td></tr>
<tr><td>101-111</td><td>Setting Prohibited</td><td colspan="4">Setting prohibited</td></tr>
</table>

Table 3-13 : VRAM-E Allocations

MST	Allocation	Address
000	ARM9	0x06880000-0x0688FFFF
001	2D Graphics Engine A	BG-VRAM (0x06000000-0x0600FFFF)
010	2D Graphics Engine A	OBJ-VRAM (0x06400000-0x0640FFFF)
011	Texture Palette	Texture palette slot 0-3
100	2D Graphics Engine A	BG extended palette slots 0-3 (only the lower 32 KB are valid)
101-111	Setting Prohibited	Setting prohibited

Table 3-14 : VRAM-F and VRAM-G Allocations

MST	Allocation / OFS	00	01	10	11
000	ARM9	VRAM-F: 0x06890000-0x06893FFF VRAM-G: 0x06894000-0x06897FFF			
001	2D Graphics Engine A	BG-VRAM (0x06000000-0x06003FFF)	BG-VRAM (0x06004000-0x06007FFF)	BG-VRAM (0x06010000-0x06013FFF)	BG-VRAM (0x06014000-0x06017FFF)
010	2D Graphics Engine A	OBJ-VRAM (0x06400000-0x06403FFF)	OBJ-VRAM (0x06404000-0x06407FFF)	OBJ-VRAM (0x06410000-0x06413FFF)	OBJ-VRAM (0x06414000-0x06417FFF)
011	Texture Palette	Slot 0	Slot 1	Slot 4	Slot 5
100	2D Graphics Engine A	BG extended palette slots 0-1	BG extended palette slots 2-3	Setting prohibited	
101	2D Graphics Engine A	OBJ extended palette slot 0 (only lower 8 KB are valid)			
110	Setting Prohibited	Setting prohibited			
111	Setting Prohibited				

Table 3-15 : VRAM-H Allocations

MST	Allocation	Address
00	ARM9	0x06898000-0x0689FFFF
01	2D Graphics Engine B	BG-VRAM (0x06200000-0x06207FFF)
10	2D Graphics Engine B	BG extended palette slots 0-3
11	Setting Prohibited	Setting prohibited

Table 3-16 : VRAM-I Allocations

MST	Allocation	Address
00	ARM9	0x068A0000-0x068A3FFFF
01	2D Graphics Engine B	BG-VRAM (0x06208000-0x0620BFFF)
10	2D Graphics Engine B	OBJ-VRAM (0x06600000-0x06603FFF)
11	2D Graphics Engine B	OBJ extended palette slots 0-1

3.2.2 Work RAM

In addition to the WRAM-0/1 (32 KB) shared by NITRO, the TWL also has the following areas of Work RAM available:

- WRAM-A[0-3] (256 KB), which can be shared between the ARM9 and ARM7
- WRAM-B[0-7] (256 KB), which can be shared between the ARM9, ARM7, and DSP
- WRAM-C[0-7] (256 KB), which can be shared between the ARM9, ARM7, and DSP

WRAM-0/1 settings are configured in the same way they were on NITRO, by setting RAM Bank Control Register 1 (WVRAMCNT).

WRAM-A, B, and C settings are configured using WRAM Bank Control Registers 1-8 (MBK1-MBK8) in the system configuration register. If the enabled regions overlap with each other, the WRAM will be mapped with the priorities WRAM-A > WRAM-B > WRAM-C. An instance and image of WRAM-01 (or ARM7-dedicated WRAM for ARM7 addresses located at or after 0x38000000) will be mapped to disabled regions.

3.2.2.1 WRAM-0/1

Work RAM does not have a fixed use, so it can be assigned for each application in ways that make the most efficient use of memory resources. This ability is called *WRAM bank control*.

Be sure not to switch banks while Work RAM is being accessed.

WVRAMCNT: RAM Bank Control Register 1

Name: WVRAMCNT Address: 0x04000244 Attribute: W Initial value: 0x00000000

31–26	25–24	23	22–21	20–19	18–16	15	14–13	12–11	10–8	7	6–3	2–0
	BANK	E		OFS	MST	E		OFS	MST	E		MST
WRAM		VRAM-G				VRAM-F				VRAM-E		

- WRAM
  - BANK[d25–d24]: Bank Specification

    Selects whether to allocate 16 KB x 2 blocks to ARM9 or ARM7.

    Figure 3-17 shows the memory maps for each value setting.

	ARM9	ARM7
00	32KB	None
01	16KB (Block1)	16KB (Block0)
10	16KB (Block0)	16KB (Block1)
11	None	32KB

Note: Though you can change the Work RAM settings, to ensure that the subprocessor API operates correctly, do not change the register settings.

Figure 3-17 : Memory Maps for Various Settings of ARM9, ARM7 Shared Internal Work RAM

ARM9 Memory Map

Allocation for Entire Region (32KB)

0x04000000	I/O registers
0x03800000	Image of ARM9, ARM7 shared internal Work RAM
0x037F8000	ARM9, ARM7 shared internal Work RAM (32KB)
0x03000000	Image of ARM9, ARM7 shared internal Work RAM
0x02000000	Main memory image

Block 1 Allocation (16KB)

0x04000000	I/O registers
0x03800000	Image of ARM9, ARM7 shared internal Work RAM (block 1)
0x037FC000	ARM9, ARM7 shared internal Work RAM (block 1: 16KB)
0x03000000	Image of ARM9, ARM7 shared internal Work RAM (block 1)
0x02000000	Main memory image

Block 0 Allocation (16KB)

0x04000000	I/O registers
0x03800000	Image of ARM9, ARM7 shared internal Work RAM (block 0)
0x037FC000	ARM9, ARM7 shared internal Work RAM (block 0: 16KB)
0x03000000	Image of ARM9, ARM7 shared internal Work RAM (block 0)
0x02000000	Main memory image

No Allocation

0x04000000	I/O registers
0x03000000	Undefined data
0x02000000	Main memory image

ARM7 Memory Map

No Allocation

0x04000000	I/O registers
0x03810000	Image of ARM7 dedicated internal Work RAM
0x03800000	ARM7 dedicated internal Work RAM (64KB)
0x03000000	Image of ARM7 dedicated internal Work RAM (32KB)
0x02000000	Main memory image

Block 0 Allocation (16KB)

0x04000000	I/O registers
0x03810000	Image of ARM7 dedicated internal Work RAM
0x03800000	ARM7 dedicated internal Work RAM (64KB)
0x037FC000	Image of ARM9, ARM7 shared internal Work RAM (block 0: 16KB)
0x03000000	Image of ARM9, ARM7 shared internal Work RAM (block 0)
0x02000000	Main memory image

Block 1 Allocation (16KB)

0x04000000	I/O registers
0x03810000	Image of ARM7 dedicated internal Work RAM
0x03800000	ARM7 dedicated internal Work RAM (64KB)
0x037FC000	Image of ARM9, ARM7 shared internal Work RAM (block 1: 16KB)
0x03000000	Image of ARM 9, ARM7 shared internal Work RAM (block 1)
0x02000000	Main memory image

Allocation for Entire Region (32KB)

0x04000000	I/O registers
0x03810000	Image of ARM7 dedicated internal Work RAM
0x03800000	ARM7 dedicated internal Work RAM (64KB)
0x037F8000	ARM9, ARM7 shared internal Work RAM (32KB)
0x03000000	Image of ARM9, ARM7 shared internal Work RAM
0x02000000	Main memory image

3.2.2.2 WRAM-A

WRAM-A can be mapped to memory by assigning four 64KB memory blocks to the ARM9 or ARM7 slots (offsets). WRAM Bank Control Register 1 (MBK1) is used to set which memory is assigned to which slot.

MBK1: WRAM Bank Control Register 1

Name: MBK1 Address: 0x04004040 Attribute: R/W Initial value: 0x00000000

31				27 26		24	23				19 18		16	15				11 10		8	7				3 2		0
E				OF		M	E				OF		M	E				OF		M	E				OF		M
WRAM-A3							WRAM-A2							WRAM-A1							WRAM-A0						

- WRAM-A[0-3]
  - E[d31][d23][d15][d07]: Enable Setting

    Selects whether to enable (1) or disable (0) each of the WRAM-A memory blocks.
  - OF[d27-d26][d19-d18][d11-d10][d03-d02]: Offset Setting

    Specifies the assignment location (offset 0-3) for each of the WRAM-A memory blocks.

	Effect of Settings
00	Assign WRAM-Ax to the offset 0 memory region (slot 0)
01	Assign WRAM-Ax to the offset 1 memory region (slot 1)
10	Assign WRAM-Ax to the offset 2 memory region (slot 2)
11	Assign WRAM-Ax to the offset 3 memory region (slot 3)

  - M[d24][d16][d08][d00]: Master Setting

    Specifies the master for each of the WRAM-A memory blocks.

	Effect of Settings
00	Set ARM9 as the master for WRAM-Ax
01	Set ARM7 as the master for WRAM-Ax

Be sure to set different combinations of masters and offsets for each WRAM-A memory block.

For example:

WRAM-A0	Slot 0 on ARM9 (M=0, OF=00)
WRAM-A1	Slot 1 on ARM9 (M=0, OF=01)
WRAM-A2	Slot 0 on ARM7 (M=1, OF=00)
WRAM-A3	Slot 1 on ARM7 (M=1, OF=01)

Next, set the region image size and the starting and ending addresses using WRAM Control Register 6 (MBK6) of the ARM9 and ARM7.

MBK6: WRAM Bank Control Register 6

Name: MBK6 Address: 0x04004054 Attribute: R/W Initial value: 0x00000000

31			28 20							13 12	11 4				0
			EADDR							IS	SADDR				
WRAM-A															

- WRAM-A
  - EADDR[d28-d20]: End Address Setting

    Specifies the end address of the WRAM-A region. The address is specified in 64KB chunks.

    End Address = 0x02FFFFFF + (EADDR << 16) (0x02FFFFFF-0x03FFFFFF)

  - IS[d13-d12]: Image Size Setting

    Specifies the image size of the WRAM-A region.

	Effect of Settings
00	Repeat the WRAM-A slot 0 image in 64KB units
01	
10	Repeat the WRAM-A slot 0-2 images in 128KB units
11	Repeat the WRAM-A slot 0-3 images in 258KB units

- Setting the image size to 64KB units (IS=00, 01)

		0x04000000	
END	**Disabled region**	0x03FF0000	Slot 0
	Enabled region	0x03030000	…
		0x03020000	Slot 0
START		0x03010000	Slot 0
	Disabled region	0x03000000	Slot 0

- Setting the image size to 128KB units (IS=10)

		0x04000000	
END	**Disabled region**	0x03FF0000	Slot 1
	Enabled region	0x03030000	…
		0x03020000	Slot 0
START		0x03010000	Slot 1

	Disabled region	0x03000000	Slot 0

- Setting the image size to 256KB units (IS=11)

		0x04000000	
END	**Disabled region**	0x03FF0000	Slot 3
	Enabled region	0x03030000	…
		0x03020000	Slot 2
START		0x03010000	Slot 1
	Disabled region	0x03000000	Slot 0

- SADDR[d11-d04]: Start Address Setting

  Specifies the start address of the WRAM-A region. The address is specified in 64KB chunks.

  Start Address = 0x03000000 + (SADDR << 16)   (0x03000000-0x03FF0000)

If enabled regions overlap, they will be mapped with the priority WRAM-A > WRAM-B > WRAM-C, and WRAM-0/1 will be mapped to any disabled regions if no other shared RAM has been mapped. Regarding access to slots to which no memory blocks have been assigned, writes are disabled, and reads will always return a 0.

For example:

If the ARM9 image size is set to 256KB units (IS=11), and the memory blocks are assigned to each slot as shown below, the slot 1 region of the ARM9 will have writes disabled, and reads will return all 0s.

WRAM-A0	Slot 0 on ARM9 (M=0, OF=00)
WRAM-A1	Slot 0 on ARM7 (M=1, OF=00)
WRAM-A2	Slot 2 on ARM9 (M=0, OF=10)
WRAM-A3	Slot 3 on ARM9 (M=0, OF=11)

3.2.2.3 WRAM-B

WRAM-B can be mapped to memory by assigning eight 32KB memory blocks to the slots (offsets) on the ARM9, ARM7, or DSP program bus. WRAM Bank Control Registers 2 and 3 (MBK2, MBK3) are used to set which memory is assigned to which slot.

MBK2: WRAM Bank Control Register 2

Name: MBK2 Address: 0x04004044 Attribute: R/W Initial value: 0x00000000

31			28	26	25	24	23			20	18	17	16	15			12	10	9	8	7			4	2	1	0
E			OF		M		E			OF		M		E			OF		M		E			OF		M	
WRAM-B3							WRAM-B2							WRAM-B1							WRAM-B0						

- WRAM-B[0-3]
  - E[d31][d23][d15][d07]: Enable Setting

    Selects whether to enable (1) or disable (0) each of the WRAM-B memory blocks.
  - OF[d28-d26][d20-d18][d12-d10][d04-d02]: Offset Setting

    Specifies the assignment location (offset 0-7) for each of the WRAM-B memory blocks.

	Effect of Settings
000	Assign WRAM-Bx to the offset 0 memory region (slot 0)
001	Assign WRAM-Bx to the offset 1 memory region (slot 1)
010	Assign WRAM-Bx to the offset 2 memory region (slot 2)
011	Assign WRAM-Bx to the offset 3 memory region (slot 3)
100	Assign WRAM-Bx to the offset 4 memory region (slot 4)
101	Assign WRAM-Bx to the offset 5 memory region (slot 5)
110	Assign WRAM-Bx to the offset 6 memory region (slot 6)
111	Assign WRAM-Bx to the offset 7 memory region (slot 7)

  - M[d25-d24][d17-d16][d09-d08][d01-d00]: Master Setting

    Specifies the master for each of the WRAM-B memory blocks.

	Effect of Settings
00	Set ARM9 as the master for WRAM-Bx
01	Set ARM7 as the master for WRAM-Bx
10	Set the DSP (program bus) as the master for WRAM-Bx
11	

MBK3: WRAM Bank Control Register 3

Name: MBK3 Address: 0x04004048 Attribute: R/W Initial value: 0x00000000

31			28	26	25	24	23			20	18	17	16	15			12	10	9	8	7			4	2	1	0
E			OF		M		E			OF		M		E			OF		M		E			OF		M	
WRAM-B7							WRAM-B6							WRAM-B5							WRAM-B4						

- WRAM-B[4-7]
  - E[d31][d23][d15][d07]: Enable Setting

    Selects whether to enable (1) or disable (0) each of the WRAM-B memory blocks.
  - OF[d28-d26][d20-d18][d12-d10][d04-d02]: Offset Setting

    Specifies the assignment location (offset 0-7) for each of the WRAM-B memory blocks.

	Effect of Settings
000	Assign WRAM-Bx to the offset 0 memory region (slot 0)
001	Assign WRAM-Bx to the offset 1 memory region (slot 1)
010	Assign WRAM-Bx to the offset 2 memory region (slot 2)
011	Assign WRAM-Bx to the offset 3 memory region (slot 3)
100	Assign WRAM-Bx to the offset 4 memory region (slot 4)
101	Assign WRAM-Bx to the offset 5 memory region (slot 5)
110	Assign WRAM-Bx to the offset 6 memory region (slot 6)
111	Assign WRAM-Bx to the offset 7 memory region (slot 7)

  - M[d25-d24][d17-d16][d09-d08][d01-d00]: Master Setting

    Specifies the master for each of the WRAM-B memory blocks.

	Effect of Settings
00	Set ARM9 as the master for WRAM-Bx
01	Set ARM7 as the master for WRAM-Bx
10	Set the DSP (program bus) as the master for WRAM-Bx
11	

Be sure to set different combinations of masters and offsets for each WRAM-B memory block.

For example:

WRAM-B0 Slot 0 on ARM9 (M=00, OF=000)
WRAM-B1 Slot 1 on ARM9 (M=00, OF=001)
WRAM-B2 Slot 2 on ARM9 (M=00, OF=010)
WRAM-B3 Slot 3 on ARM9 (M=00, OF=011)

WRAM-B4	Slot 0 on ARM7 (M=01, OF=000)
WRAM-B5	Slot 1 on ARM7 (M=01, OF=001)
WRAM-B6	Slot 0 on DSP (program bus) (M=10, OF=000)
WRAM-B7	Slot 1 on DSP (program bus) (M=10, OF=001)

Next, set the region image size and the starting and ending addresses using WRAM Control Register 7 (MBK7) of the ARM9 and ARM7.

MBK7: WRAM Bank Control Register 7

Name: MBK7 Address: 0x04004058 Attribute: R/W Initial value: 0x00000000

31–29	28–19	18–14	13–12	11–3	2–0
	EADDR		IS	SADDR	
WRAM-B					

- WRAM-B
  - EADDR[d28-d19]: End Address Setting

    Specifies the end address of the WRAM-B region. The address is specified in 32KB chunks.

    End Address = 0x02FFFFFF + (EADDR << 15) (0x02FFFFFF-0x03FFFFFF)

  - IS[d13-d12]: Image Size Setting

    Specifies the image size of the WRAM-B region.

	Effect of Settings
00	Repeat the WRAM-B slot 0 image in 32KB units
01	Repeat the WRAM-B slot 0-1 images in 64KB units
10	Repeat the WRAM-B slot 0-3 images in 128KB units
11	Repeat the WRAM-B slot 0-7 images in 258KB units

- Setting the image size to 32KB units (IS=00)

0x04000000

	Disabled region	0x03FF8000	Slot 0
END		0x03FF0000	Slot 0
	Enabled region	0x03030000	...
		0x03028000	Slot 0
		0x03020000	Slot 0
		0x03018000	Slot 0
START		0x03010000	Slot 0
	Disabled region	0x03008000	Slot 0
		0x03000000	Slot 0

- Setting the image size to 64KB units (IS=01)

		0x04000000	
	Disabled region	0x03FF8000	Slot 1
END		0x03FF0000	Slot 0
	Enabled region	0x03030000	...
		0x03028000	Slot 1
		0x03020000	Slot 0
		0x03018000	Slot 1
START		0x03010000	Slot 0
	Disabled region	0x03008000	Slot 1
		0x03000000	Slot 0

- Setting the image size to 128KB units (IS=10)

		0x04000000	
	Disabled region	0x03FF8000	Slot 3
END		0x03FF0000	Slot 2
	Enabled region	0x03030000	...
		0x03028000	Slot 1
		0x03020000	Slot 0
		0x03018000	Slot 3
START		0x03010000	Slot 2
	Disabled region	0x03008000	Slot 1
		0x03000000	Slot 0

- Setting the image size to 256KB units (IS=11)

		0x04000000	
	Disabled region	0x03FF8000	Slot 7
END		0x03FF0000	Slot 6
	Enabled region	0x03030000	...
		0x03028000	Slot 5
		0x03020000	Slot 4
		0x03018000	Slot 3
START		0x03010000	Slot 2
	Disabled region	0x03008000	Slot 1
		0x03000000	Slot 0

- SADDR[d11-d03]: Start Address Setting

  Specifies the start address of the WRAM-B region. The address is specified in 32KB chunks.

  Start Address = 0x03000000 + (SADDR << 15) (0x03000000-0x03FF8000)

If enabled regions overlap, they will be mapped with the priority WRAM-A > WRAM-B > WRAM-C, and WRAM-0/1 will be mapped to any disabled regions if no other shared RAM has been mapped. Regarding access to slots to which no memory blocks have been assigned, writes are disabled, and reads will always return a 0.

3.2.2.4 WRAM-C

WRAM-C can be mapped to memory by assigning eight 32KB memory blocks to the slots (offsets) on the ARM9, ARM7, or DSP data bus. WRAM Bank Control Registers 4 and 5 (MBK4, MBK5) are used to set which memory is assigned to which slot.

MBK4: WRAM Bank Control Register 4

Name: MBK4 Address: 0x0400404C Attribute: R/W Initial value: 0x00000000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
E			OF			M		E			OF			M		E			OF			M		E			OF			M	
WRAM-C3								WRAM-C2								WRAM-C1								WRAM-C0							

- WRAM-C[0-3]
  - E[d31][d23][d15][d07]: Enable Setting

    Selects whether to enable (1) or disable (0) each of the WRAM-C memory blocks.
  - OF[d28-d26][d20-d18][d12-d10][d04-d02]: Offset Setting

    Specifies the assignment location (offset 0-7) for each of the WRAM-C memory blocks.

	Effect of Settings
000	Assign WRAM-Cx to the offset 0 memory region (slot 0)
001	Assign WRAM-Cx to the offset 1 memory region (slot 1)
010	Assign WRAM-Cx to the offset 2 memory region (slot 2)
011	Assign WRAM-Cx to the offset 3 memory region (slot 3)
100	Assign WRAM-Cx to the offset 4 memory region (slot 4)
101	Assign WRAM-Cx to the offset 5 memory region (slot 5)
110	Assign WRAM-Cx to the offset 6 memory region (slot 6)
111	Assign WRAM-Cx to the offset 7 memory region (slot 7)

- M[d25-d24][d17-d16][d09-d08][d01-d00]: Master Setting

  Specifies the master for each of the WRAM-C memory blocks.

	Effect of Settings
00	Set ARM9 as the master for WRAM-Cx
01	Set ARM7 as the master for WRAM-Cx
10	Set the DSP (data bus) as the master for WRAM-Cx
11	Set the DSP (data bus) as the master for WRAM-Cx

MBK5: WRAM Bank Control Register 5

Name: MBK5 Address: 0x04004050 Attribute: R/W Initial value: 0x00000000

<table>
<tr><td>31</td><td></td><td></td><td colspan="3">28 26</td><td>25</td><td>24</td><td>23</td><td></td><td></td><td colspan="3">20 18</td><td>17</td><td>16</td><td>15</td><td></td><td></td><td colspan="3">12 10</td><td>9</td><td>8</td><td>7</td><td></td><td></td><td colspan="3">4 2</td><td>1</td><td>0</td></tr>
<tr><td>E</td><td></td><td></td><td colspan="3">OF</td><td colspan="2">M</td><td>E</td><td></td><td></td><td colspan="3">OF</td><td colspan="2">M</td><td>E</td><td></td><td></td><td colspan="3">OF</td><td colspan="2">M</td><td>E</td><td></td><td></td><td colspan="3">OF</td><td colspan="2">M</td></tr>
<tr><td colspan="8">WRAM-C7</td><td colspan="8">WRAM-C6</td><td colspan="8">WRAM-C5</td><td colspan="8">WRAM-C4</td></tr>
</table>

- WRAM-C[4-7]
  - E[d31][d23][d15][d07]: Enable Setting

    Selects whether to enable (1) or disable (0) each of the WRAM-C memory blocks.
  - OF[d28-d26][d20-d18][d12-d10][d04-d02]: Offset Setting

    Specifies the assignment location (offset 0-7) for each of the WRAM-C memory blocks.

	Effect of Settings
000	Assign WRAM-Cx to the offset 0 memory region (slot 0)
001	Assign WRAM-Cx to the offset 1 memory region (slot 1)
010	Assign WRAM-Cx to the offset 2 memory region (slot 2)
011	Assign WRAM-Cx to the offset 3 memory region (slot 3)
100	Assign WRAM-Cx to the offset 4 memory region (slot 4)
101	Assign WRAM-Cx to the offset 5 memory region (slot 5)
110	Assign WRAM-Cx to the offset 6 memory region (slot 6)
111	Assign WRAM-Cx to the offset 7 memory region (slot 7)

- M[d25-d24][d17-d16][d09-d08][d01-d00]: Master Setting

  Specifies the master for each of the WRAM-C memory blocks.

	Effect of Settings
00	Set ARM9 as the master for WRAM-Cx
01	Set ARM7 as the master for WRAM-Cx
10	Set the DSP (data bus) as the master for WRAM-Cx
11	

Be sure to set different combinations of masters and offsets for each WRAM-B memory block.

For example:

WRAM-C0	Slot 0 on ARM9 (M=00, OF=000)
WRAM-C1	Slot 1 on ARM9 (M=00, OF=001)
WRAM-C2	Slot 0 on ARM7 (M=01, OF=000)
WRAM-C3	Slot 1 on ARM7 (M=01, OF=001)
WRAM-C4	Slot 0 on DSP (data bus) (M=10, OF=000)
WRAM-C5	Slot 1 on DSP (data bus) (M=10, OF=001)
WRAM-C6	Slot 2 on DSP (data bus) (M=10, OF=010)
WRAM-C7	Slot 3 on DSP (data bus) (M=10, OF=011)

Next, set the region image size and the starting and ending addresses using WRAM Control Register 8 (MBK8) of the ARM9 and ARM7.

MBK8: WRAM Bank Control Register 8

Name: MBK8 Address: 0x0400405C Attribute: R/W Initial value: 0x00000000

31	28	19		13	12	11	3	0
	EADDR			IS		SADDR		
WRAM-C								

- WRAM-C
  - EADDR[d28-d19]: End Address Setting

    Specifies the end address of the WRAM-C region. The address is specified in 32KB chunks.

    End Address = 0x02FFFFFF + (EADDR << 15) (0x02FFFFFF-0x03FFFFFF)

  - IS[d13-d12]: Image Size Setting

    Specifies the image size of the WRAM-C region.

	Effect of Settings
00	Repeat the WRAM-C slot 0 image in 32KB units
01	Repeat the WRAM-C slot 0-1 images in 64KB units
10	Repeat the WRAM-C slot 0-3 images in 128KB units
11	Repeat the WRAM-C slot 0-7 images in 256KB units

- Setting the image size to 32KB units (IS=00)

0x04000000

	Disabled region	0x03FF8000	Slot 0
END		0x03FF0000	Slot 0
	Enabled region	0x03030000	Åc
		0x03028000	Slot 0
		0x03020000	Slot 0
		0x03018000	Slot 0
START		0x03010000	Slot 0
	Disabled region	0x03008000	Slot 0
		0x03000000	Slot 0

- Setting the image size to 64KB units (IS=01)

		0x04000000	
	Disabled region	0x03FF8000	Slot 1
END		0x03FF0000	Slot 0
	Enabled region	0x03030000	Åc
		0x03028000	Slot 1
		0x03020000	Slot 0
		0x03018000	Slot 1
START		0x03010000	Slot 0
	Disabled region	0x03008000	Slot 1
		0x03000000	Slot 0

- Setting the image size to 128KB units (IS=10)

		0x04000000	
	Disabled region	0x03FF8000	Slot 3
END		0x03FF0000	Slot 2
	Enabled region	0x03030000	Åc
		0x03028000	Slot 1
		0x03020000	Slot 0
		0x03018000	Slot 3
START		0x03010000	Slot 2
	Disabled region	0x03008000	Slot 1
		0x03000000	Slot 0

- Setting the image size to 256KB units (IS=11)

		0x04000000	
	Disabled region	0x03FF8000	Slot 7
END		0x03FF0000	Slot 6
	Enabled region	0x03030000	Åc
		0x03028000	Slot 5
		0x03020000	Slot 4
		0x03018000	Slot 3
START		0x03010000	Slot 2
	Disabled region	0x03008000	Slot 1
		0x03000000	Slot 0

TWL-06-0017-001-D
Released: February 16, 2009

- SADDR[d11-d03]: Start Address Setting

  Specifies the start address of the WRAM-C region. The address is specified in 32KB chunks.

  Start Address = 0x03000000 + (SADDR << 15)   (0x03000000-0x03FF8000)

If enabled regions overlap, they will be mapped with the priority WRAM-A > WRAM-B > WRAM-C, and WRAM-0/1 will be mapped to any disabled regions if no other shared RAM has been mapped. For access to slots to which no memory blocks have been assigned, writes are disabled, and reads will always return a 0.

3.2.3 Work RAM Configuration in TWL SDK

With TWL SDK, the WRAM manager in the memory interface library handles the arbitration of requests to assign WRAM-A/B/C to each processor.

WRAM-A (64 KB x 4) is completely used by the ARM7, so its allocation cannot be changed. In contrast, the allocation of WRAM-B (32 KB x 8) and WRAM-C (32 KB x 8) can be changed flexibly between ARM9, ARM7, and the DSP.

For details about the WRAM manager, refer to “Work RAM” in the Memory Interface Overview section of the *TWL SDK Function Reference Manual*.

3.2.4 I/O Registers

See the appendices to learn about the mapping for each I/O register.

- Accessing undefined registers

  Table 3-17 shows the behaviors that occur when an undefined address is accessed in the I/O register regions.

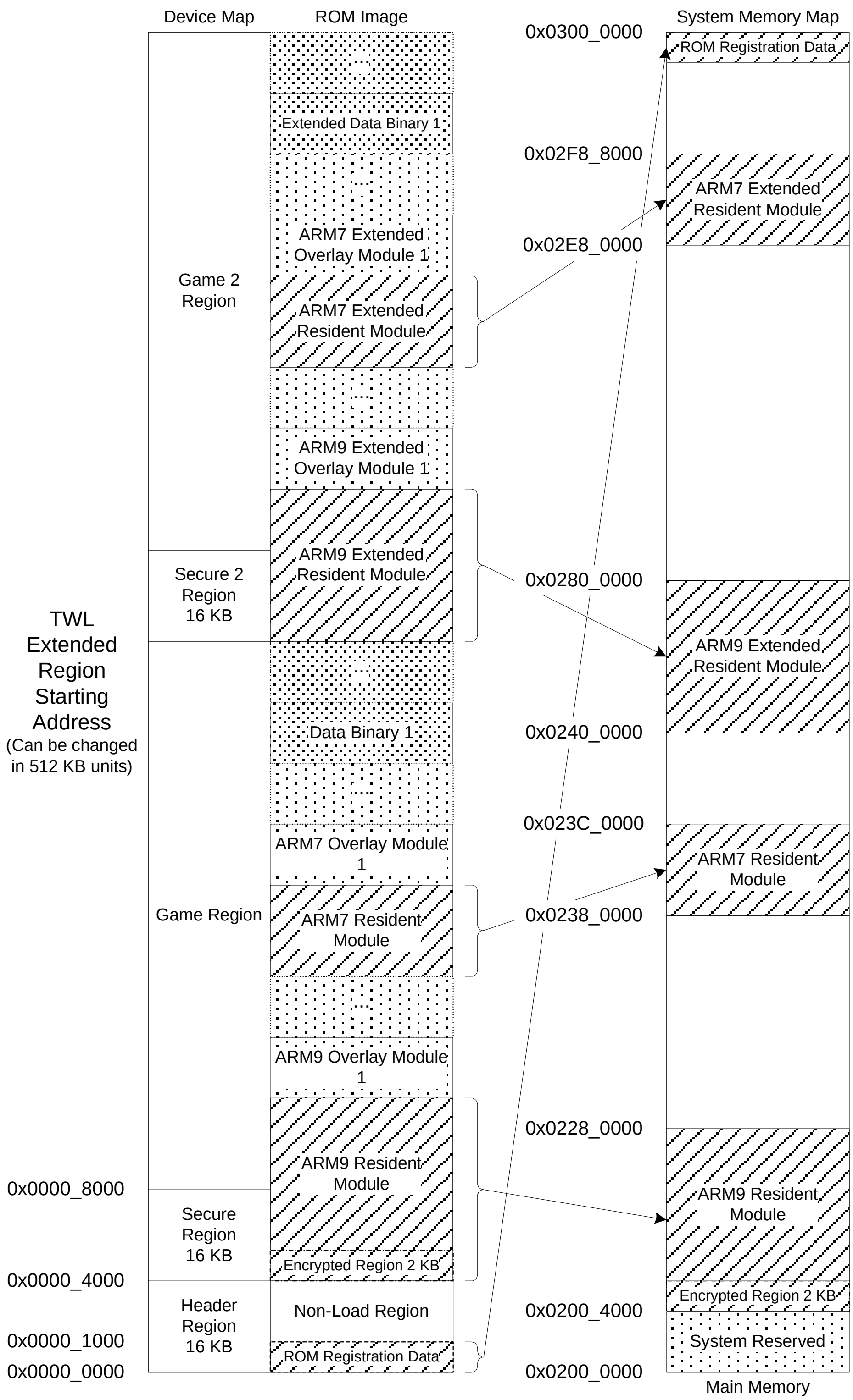
Table 3-17 : Result of Accessing an Undefined Register

Access Destination	Write	Read
Undefined Register	Invalid	ALL zero

3.3 Memory Map for Game Card Boot

Figure 3-18 shows the memory map when the system boots from a Game Card.

Figure 3-18 : Memory Map for Game Card Boot



The following are details about the regions shown in the memory map. Regions whose names include "Extended," the Secure 2 Region, and the Game 2 Region have been extended in TWL.

In Nitro compatibility mode, only 4 MB from the start of the main memory is enabled. There is no system reserved region at the start, and the ARM9 resident module can be located at the start of main memory.

- Resident Module, Extended Resident Module:

  Loaded at application startup time.
- Overlay Module, Extended Overlay Module:

  Loaded by the application itself.
- Data Binary, Extended Data Binary:

  Loaded by the application itself.

  Not linked; loaded by referencing FAT.
- The Header Region:

  Stores the ROM registration data. This data is loaded into the System region of main memory at boot time.

  This region cannot be reloaded after application startup.

  (After the Game Code is determined) Provides a binary file called a ROM header template for each application.
- The Secure Region:

  Stores the starting part of the ARM9 Resident Module. This is loaded at boot time to the address in main memory specified by the registration data in ROM.

  Must be located within the first 64 KB from the start of main memory, for security reasons.

  Decodes the first 2 KB, an encrypted region, when loaded.

  To be specific, the encrypted system call library is linked to the start when the ARM9 resident module is created.

  Because this cannot be reloaded after application startup, only the `.text` and `.rodata` sections should be located in the secure region. Otherwise, you cannot perform a software reset.
- The Secure 2 Region:

  Stores the starting part of the ARM9 Extended Resident Module. This is loaded at boot time to the address in main memory specified by the registration data in ROM.

  Like the Secure Region, it cannot be reloaded after application startup.
- The Game Region:

  Loads (at boot time) the part of the ARM9 Resident Module that follows the Secure Region and the ARM7 Resident Module to the address specified by the registration data in ROM.

  Set the bootable size for ARM9 so that it does not exceed 2.5 MB from the beginning of main memory (note that 16 KB will be subtracted for the System Reserved region). For ARM7, the bootable region should not exceed 256 KB from address 0x02380000 in main memory.

  This region is always read-enabled. The application should load the Overlay Module and Data Binary as needed.
- The Game 2 Region:

  Loads (at boot time) the part of the ARM9 Extended Resident Module that follows the Secure Region and the ARM7 Extended Resident Module to the address specified by the registration data in ROM.

  Set the bootable size for ARM9 so that it does not exceed 4 MB from address 0x02400000 in main memory. For ARM7, the bootable region should not exceed 1.03 KB starting from address 0x02E80000 in main memory.

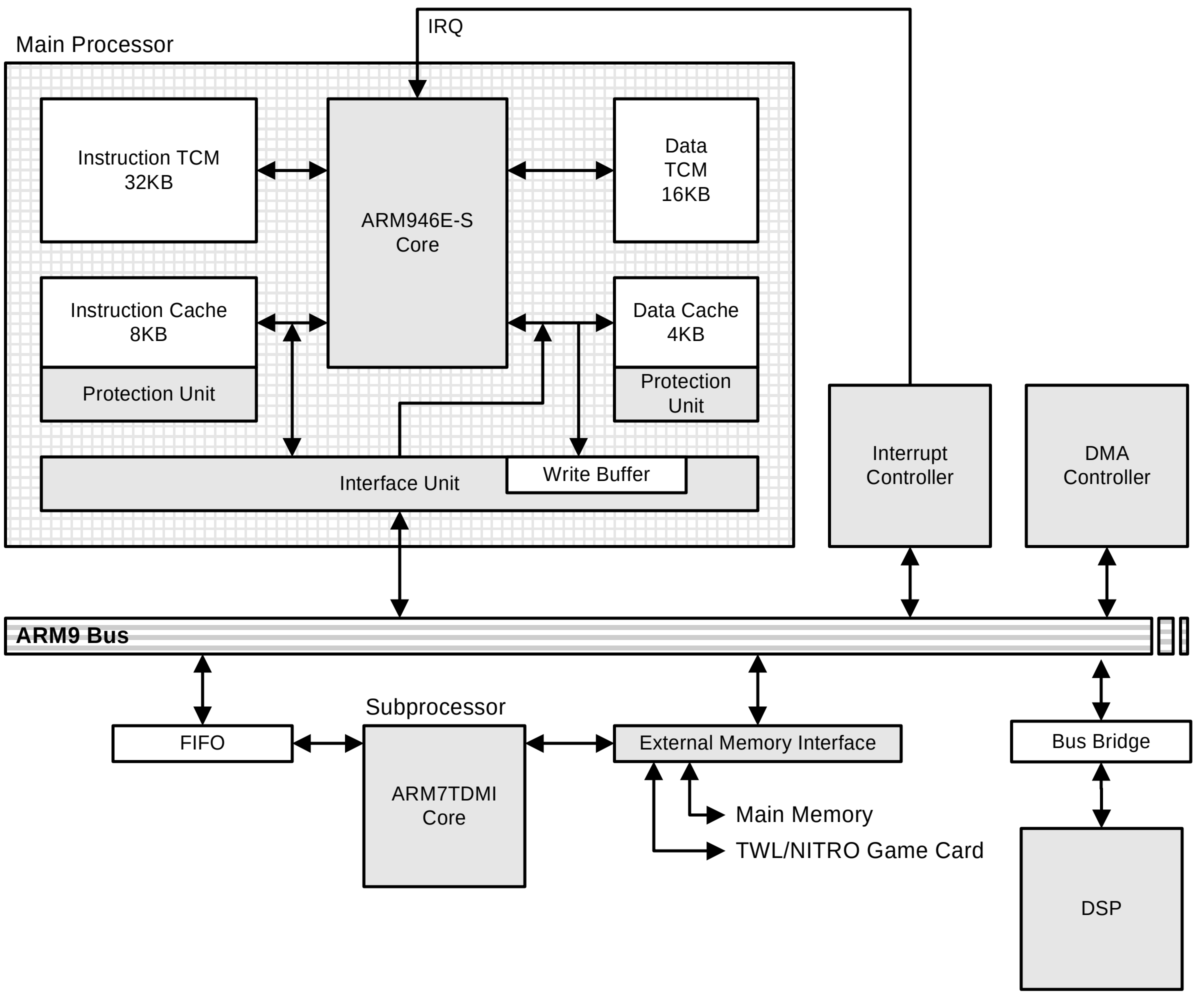
This region is always read-enabled. The application should load the Extended Overlay Module and Extended Data Binary as needed.

Note: See the *TWL Nintendo DS/DSi-Compatible/DSi Game Card Manual* for details about the ROM Registration Data.

4 Main Processor Core (ARM946E-S)

Figure 4-1 is a block diagram of the main processor core.

Figure 4-1 : Block Diagram of the Main Processor Core



4.1 Protection Unit

The *Protection Unit* protects memory by configuring the read/write attributes and enabling/disabling the use of the cache and Write buffers in each protection region.

Up to eight memory regions can be configured from the 4 GB of background (the entire address space accessible from the CPU). Protection regions with higher region numbers have higher priority. (If regions have duplicate protection region settings, the setting with the higher protection region number is given priority.) Each protection region can be configured separately to enable/disable the cache and Write buffers and allow/disallow reading/writing.

- **Debug Version and Release Version**

  In addition to the production version of TWL, there is a debug version that comes with the main memory expanded to 32 MB. Though part of the additional 16 MB of main memory is used as the debugger region, the remainder can be used during application development. Because the debug and release versions of TWL have different memory structures, the Protection Unit settings are also different.

4.2 Tightly Coupled Memory (TCM)

TCM is high-speed memory that is directly connected to the ARM9 core. It can be used as independent Work RAM. Because TCM does not use the ARM9 bus, ARM9 can use TCM to conduct processes even while the ARM9 bus is being used (such as by DMA).

As a result, TCM can boost performance by storing frequently used programs and data that you want to access during DMA transfers.

There are two types of TCM: one for instructions and one for data. Note that TCM cannot be accessed via DMA.

4.2.1 Instruction TCM

This is 32 KB of high-speed memory, mapped from ARM9 address 0x01FF8000.

Instruction TCM is a good place to hold fast-running programs or programs for which you want to fix the operation clock count. Examples include graphic libraries and routines that branch on interrupts.

Because Instruction TCM does not require the ARM9 bus while fetching instructions, Instruction TCM is also an effective place to store programs you want to execute while DMA is using the ARM9 bus. Examples include routines that generate display lists and computation routines.

You can store data in Instruction TCM memory also, but stalls occur when data access collides with instruction fetches.

4.2.2 Data TCM

This is 16 KB of high-speed memory. Data TCM can be set to any address location in the memory map.

Data TCM is a good place to store data for fast reading and writing. Examples include stacks and frequently accessed tables.

Because Data TCM does not require the ARM9 bus during access, Data TCM can be used for creating the next display list even while the current display list is being transferred via DMA from main memory to the geometry engine.

However, transfers from TCM must be conducted via the ARM9 core. In addition, instructions cannot be placed in Data TCM. But that means there is no stalling of data access due to collisions with instruction fetches.

4.3 Cache Memory

When the ARM9 bus references memory, it takes 32 bytes of data from that vicinity (the data in a range corresponding to the upper 27 bits of the referenced address) and loads that data into the cache so the data is accessible at the fast speeds of cache memory the next time that range is referenced.

If the ARM9 hits data that is in the cache, the data can be read quickly.

If the ARM9 does not hit data in the cache, the contents of memory are read in units of cache lines (line fetching). During this process, the contents of the cache lines are replaced according to the replacement algorithm.

Note that the Protection Unit must be enabled to use the cache.

Table 4-1 lists the specifications for the cache.

Table 4-1 : Cache Specifications

Capacity	Instruction cache: 8 KB Data cache: 4 KB
Configuration	4-way set associative method
Cache Line	8 words (32 bytes)
Write-miss Operation	Read-allocate method (see note 1)
Replacement Algorithm	Can choose either round-robin (see note 2) or pseudo-random (see note 3)
Bus Snoop Feature (see note 4)	None
Other Features (Instruction Cache)	Lockdown Instruction prefetch
Other Features (Data Cache)	Lockdown Write-through mode / Write-back mode

Note 1: Read-allocate methods

In this method, when there is a write miss, the write is performed only for the memory (or for the Write buffer, if it is enabled). Data is not loaded to the cache. (For the Write-allocation method, a write miss is handled as a write hit after data is loaded into the cache.)

Note 2: Round-robin (recommended)

In this method, cache lines are replaced in order. Performance is stable in the worst-case scenario.

Note 3: Pseudo-random

In this method, cache lines are replaced randomly. This increases peak-time performance, but lowers worst-case performance.

Note 4: Bus Snoop feature

By monitoring the bus, this feature detects whether another processor or DMA writes to the memory region stored in the cache. Because the ARM946E-S does not have this feature, you must be careful that there is coherency between memory and cache. (See "4.5 Ensuring Coherency" on page 67.)

4.3.1 Instruction Cache

This is 8 KB of high-speed memory, dedicated to instruction code.

The ARM9 bus is not used during cache hits, so a program in the Instruction cache can run even when a non-ARM9 bus master (DMA or subprocessor) has possession of the ARM9 bus.

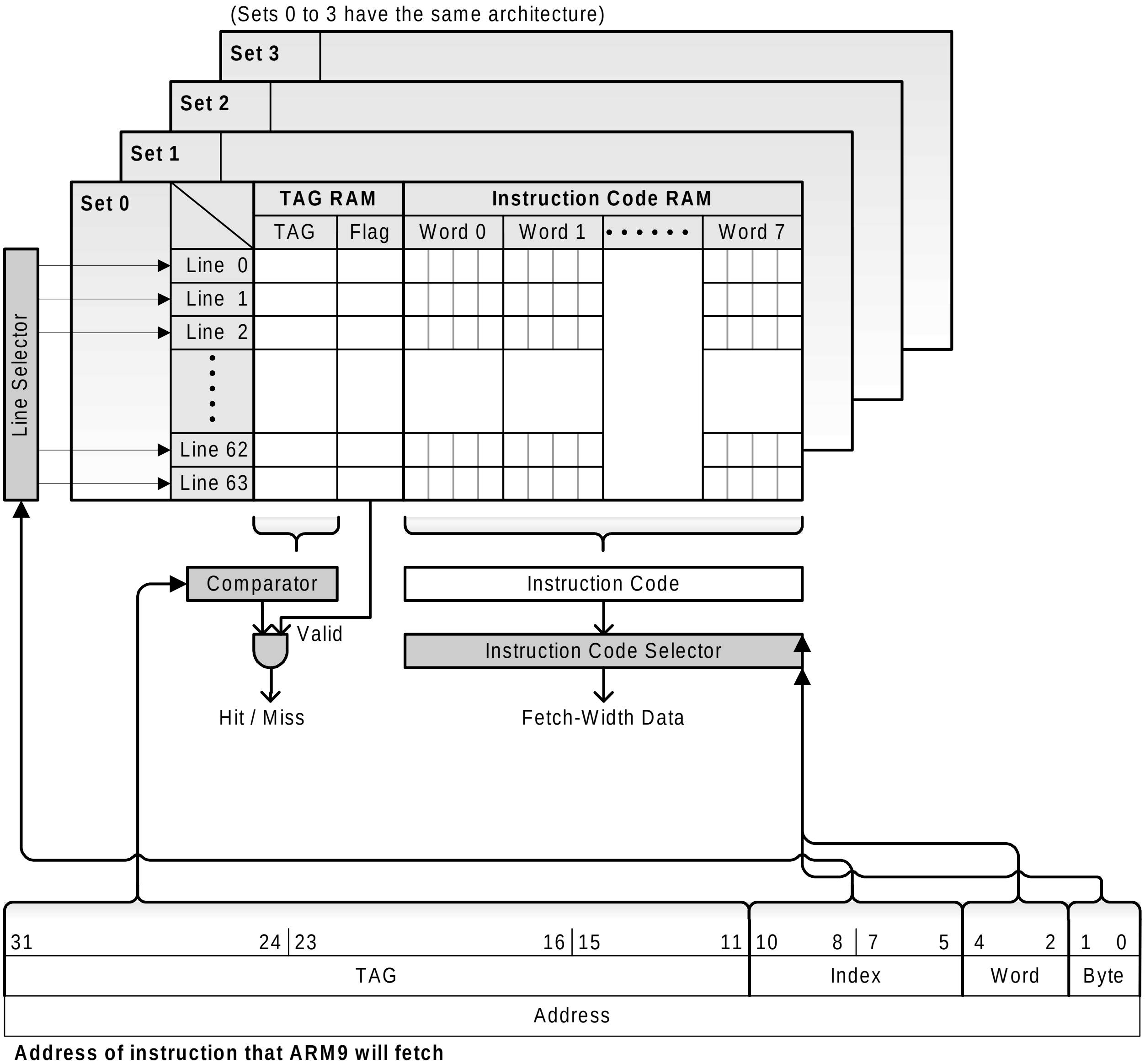
4.3.1.1 Determining Hits and Misses

When the ARM9 bus fetches instruction code from memory, the Instruction Cache Controller extracts the Index bit and TAG bit from the memory addresses and compares the contents of TAG RAM in the *n*th cache line (defined by the Index number) with the TAG bit at that memory address. Because the ARM946E-S cache has a four-set structure, the comparison is made on four cache lines. If the comparison finds a match for any of these four cache lines and the valid bit is enabled, it is determined that the cache line contains the targeted instruction code (a hit). If not, it is considered a miss.

For a cache hit, the intended instruction code in Data RAM is identified from the address' Word or Byte and can be accessed quickly. For a miss, the cache lines are fetched from memory via the ARM9 bus.

Figure 4-2 shows the structure and actions of the Instruction cache.

Figure 4-2 : Structure and Actions of the Instruction Cache



About Instruction Cache TAG RAM

- TAG

  The upper 21 bits of the memory address of the data in the cache line are stored here.

- Flag

  Generated from the *valid* bit, which indicates whether cache lines are enabled or disabled.

4.3.2 Data Cache

This is 4 KB of high-speed memory. It is data-only.

The ARM9 bus is not used during cache hits, so the data in this cache can be accessed even when a non-ARM9 bus master (DMA or subprocessor) has possession of the ARM9 bus.

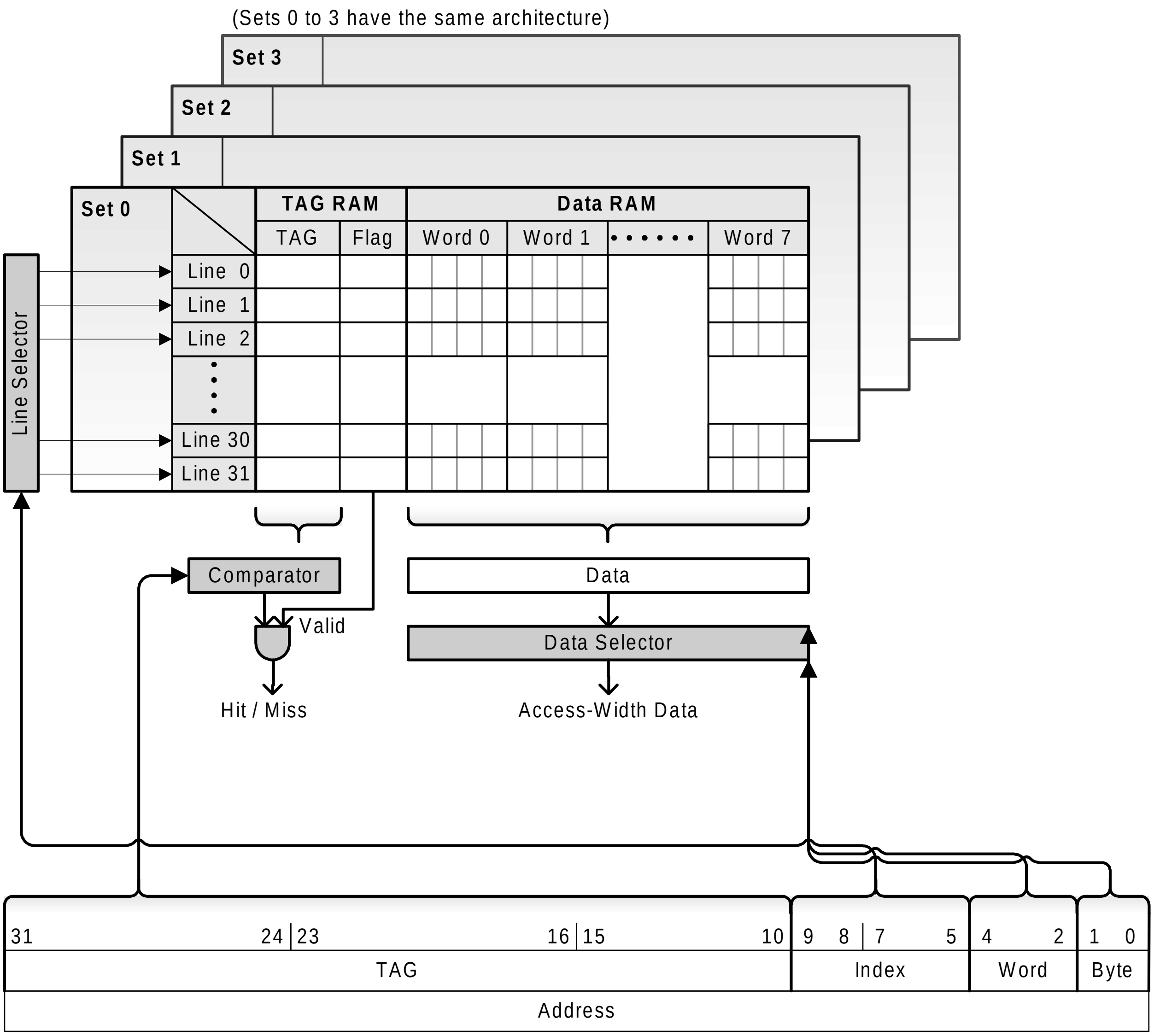
4.3.2.1 Determining Hits and Misses

When ARM9 loads data into memory, the Data Cache Controller extracts the Index bit and TAG bit from the memory address and compares the contents of TAG RAM in the *n*th cache line (defined by the index number) with the TAG bit at the memory address. The ARM946E-S cache has a four-set structure, so the comparison is made on four cache lines. If the comparison finds a match for any of these four cache lines and the valid bit is enabled, it is determined that the cache line contains the targeted data (hit). If not, it is considered a miss.

For a cache hit, the targeted data in data RAM is identified from the address' Word and Byte and can be quickly accessed. For a miss, the data is accessed from memory via the ARM9 bus.

Figure 4-3 shows the structure and actions of the Data cache.

Figure 4-3 : Structure and Actions of the Data Cache



Address of data that ARM9 will read/write
(Differs from Instruction cache in that the Index is 5 bits)

About Data Cache TAG RAM

- TAG

  The upper 22 bits of the memory address of the data in the cache line are stored here.

- Flag

  Consists of the valid bit, which indicates whether cache lines are enabled or disabled, the dirty bit for the first half of the line, and the dirty bit for the second half of the line, which indicate whether the contents of the cache and memory are the same.

  When the contents of the cache match those in memory, the cache is said to be *clean*. When the contents do not match, the cache is *dirty*.

  The dirty bits are used only in write-back mode. To read more about this mode, see "4.4 Write Buffer" on page 66.

  The dirty bits are referenced when the contents of a line are to be written back to memory. If the flag is dirty, a write-back occurs. If the flag is clean, no write-back occurs (because it is unnecessary).

4.3.3 Cache Operations

Table 4-2 shows the operations conducted on the cache. Some operations act on the entire cache, while other operations act on specified cache sets, specified cache lines, or a particular cache line specified by a memory address.

Table 4-2 : Cache Operations

Cache Operation	Instruction Cache	Data Cache
Overall Direct Operations	- Enable/Disable - Invalidate	- Enable/Disable - Invalidate
Direct Operations on Sets	- Lockdown	- Lockdown
Direct Operations on Lines	---	- Clean - Clean & Invalidate
Operations on Cache Lines Corresponding to Memory Addresses	- Prefetch - Invalidate	- Clean - Invalidate - Clean & Invalidate

The *Clean*, *Invalidate*, and *Clean & Invalidate* operations can cause changes in the usage state of cache. To read more about these changes, see "4.5 Ensuring Coherency" on page 67.

About Each Operation

- Clean

  Writes back dirty data in the cache to memory. The data remains in the cache. If the Write buffer is enabled, the actual write back to memory is delayed.

- Invalidate

  Invalidates the data in the cache. The next time this memory region is read, a read miss occurs and a line fetch from memory to the cache is performed. The data moved into the cache by the line fetch is treated as valid.

- Clean & Invalidate

  Writes back dirty data in the cache to memory and invalidates the data in the cache. As a result, the next time this memory region is accessed, the data will be line-fetched from memory to the cache.

  Caution: If the write buffer (refer to "4.4 Write Buffer" on page 66 for details) is full and the Clean & Invalidate command is issued, caches that are already clean will not be invalidated.

- Instruction Prefetch (prefetch)

  Takes instruction code that has not yet been fetched by the program and preloads it into the Instruction cache. It uses coprocessor instructions.

- Lockdown

  By locking down one set of cache, no line in that set can be replaced by line fetches and the whole set can be used as a block of Work RAM. However, this reduces the cache region by the same amount and increases the miss rate.

- Enable/Disable

  When using the cache, in addition to enabling the cache, you must also enable the Protection Unit.

You can enable/disable the Instruction cache and the Data cache for each protection region. To read about protection regions and their settings, see "4.1 Protection Unit" on page 59.

Caution: Data Preload (preload) is a feature that preloads data that has not been accessed by the program and normally operates by using the PLD instruction. However, due to ARM946E-S specifications, no operation will be performed even when a PLD instruction is recognized. Therefore, the data preload feature does not work with ARM946E-S.

4.3.4 Optimizing the Cache

The cache uses the memory reference locality of most programs to accelerate memory access.

Temporal locality	The high probability that data, once referenced, will be referenced again soon.
Spatial locality	The high probability that data near referenced data will also be referenced.

Programs with a higher reference locality have higher cache hit rates and faster average access speeds.

Depending on how a program is pieced together, you can manually increase the reference locality to some extent. For example, when constructing a loop to handle a two-dimensional (or higher) array, you can boost the spatial locality of the array by handling addresses in consecutive order.

Examples:

Loop Example A	Loop Example B
`u32 RESULT, TEST[0x100][0x100];` `for( j=0; j<0x100; j++) {` `    for( i=0; i<0x100; i++) {` `        RESULT += TEST[i][j];` `    }` `}`	`u32 RESULT, TEST[0x100][0x100];` `for( i=0; i<0x100; i++) {` `    for( j=0; j<0x100; j++) {` `        RESULT += TEST[i][j];` `    }` `}`

In loop example A, the TEST array is referenced every 0x100 addresses inside the loop, so there is no cache hit during the first iteration. Lines are fetched one after another in the first iteration. However, because the capacity of the Data cache is 32 lines x 4 sets for a total of 128 lines (0x80 lines), the entire loop does not fit in the Data cache, and hits occur only half the time in the second and subsequent iterations.

In contrast, in loop example B, the TEST array is referenced in the same order as the addresses inside the loop, enabling the maximum hit rate.

4.4 Write Buffer

The Write buffer is 32 bytes x 16 layers of FIFO memory that integrates addresses and data.

The type of entry is determined by the address/data flag. Address entries have an appended data size.

Writing data to the high-speed Write buffer instead of memory keeps ARM9 from stalling during the write process. However, ARM9 will stall when writing to the Write buffer when it is full.

When the Protection Unit is enabled, you can select among the access modes shown in Table 4-3 for writing data. The selection is made by configuring the Data cache and the Write buffer for each protection region as shown.

To read more about protection regions, see "4.1 Protection Unit" on page 59.

Table 4-3 : Access Modes When the Data is Being Written

Data Cache Setting	Write Buffer Setting	Access Mode
Disable	Disable	NCNB mode[a] (Data cache and Write buffer are both disabled)
Disable	Enable	NCB mode (Data cache disabled; Write buffer enabled)
Enable	Disable[b]	Write-through mode
Enable	Enable	Write-back mode

a. In NCNB mode, the contents of the write buffer are output when writing and are accessed ahead of the Write buffer output when reading.
b. In write-through mode, the Write buffer is disabled, but it is used.

- Write-back mode (recommended)

  If there is a hit during the data write, data is written only to the cache and not to the Write buffer. Therefore, the contents of the cache may not be the same as in memory, but writing is fast.

  A *Clean* operation must be performed in order for the data rewritten by the CPU to be reflected in memory.

  Further, when a read-miss occurs while the cache is full and the cache lines to be emptied by the replacement algorithm are dirty, they are written back to the Write buffer.

- Write-through mode

  If there is a hit during the data write, the data is written to the Write buffer at the same time that it is written to the cache. Therefore, the cache line does not become dirty as a result of writing to ARM9, but writing is slow.

  Further, if the cache is full when a read-miss occurs, the replacement algorithm overwrites the cache line.

In both of these modes, data is written only to the Write buffer when a write-miss occurs.

Also, if a line is being fetched, the contents of the Write buffer are discharged first to maintain data coherency.

Note: Be careful about the access width of memory when writing in write-through mode. (For example, you cannot use an access width of 8 bits with VRAM.) To read about the access width for each memory type, see "3 Memory" on page 17.

To read about cache state transitions and control in either of these modes, see "4.5 Ensuring Coherency" on page 67.

4.4.1 Write Buffer Operations

- Wait for the Write buffer to empty

  The ARM9 bus can stall until all data in the Write buffer is written to memory.

  Use this operation to make certain that all content rewritten by the CPU is reflected in memory.

  Note that this operation is not necessary when data is written to I/O registers or other regions where cache and buffers are disabled, because in these cases, the CPU stops until the Write buffer is empty.

  To read why this operation is necessary in other cases, see "4.5 Ensuring Coherency" on page 67.

4.5 Ensuring Coherency

Be careful when using the cache to make sure no inconsistencies arise between the contents of the cache and memory.

The cache state is managed in each cache line by the flag in TAG RAM.

This flag includes one *valid* bit and two *dirty* bits. The dirty bits indicate the state of the first half and second half of the cache line. All three bits are used for the Data cache in the write-back mode, but only the valid bit is used in the write-through mode or for the Instruction cache.

4.5.1 Write-Back Mode

Table 4-4 shows how the flag states define the cache line state in write-back mode.

Table 4-4 : Cache Line States (Write-Back Mode)

State	Flag State		Description
	Valid	Dirty	
Dirty	**1**	**1**	Cache line is valid, contents differ from memory
Clean	**1**	**0**	Cache line is valid, contents match memory
Invalid	**0**	*****	Cache line is invalid

Operations Managed Automatically by the Cache Controller

As shown in Figure 4-4, read misses/hits and write misses/hits on access from the ARM9 as well as state transitions by the replacement algorithm are performed automatically.

When the replacement algorithm replaces valid but dirty lines, the lines are first written back to memory (or to the Write buffer, if enabled). Because the process is actually conducted on the first and second halves of the line, and not on the entire line, the volume of data written back can be 0, 16, or 32 bytes.

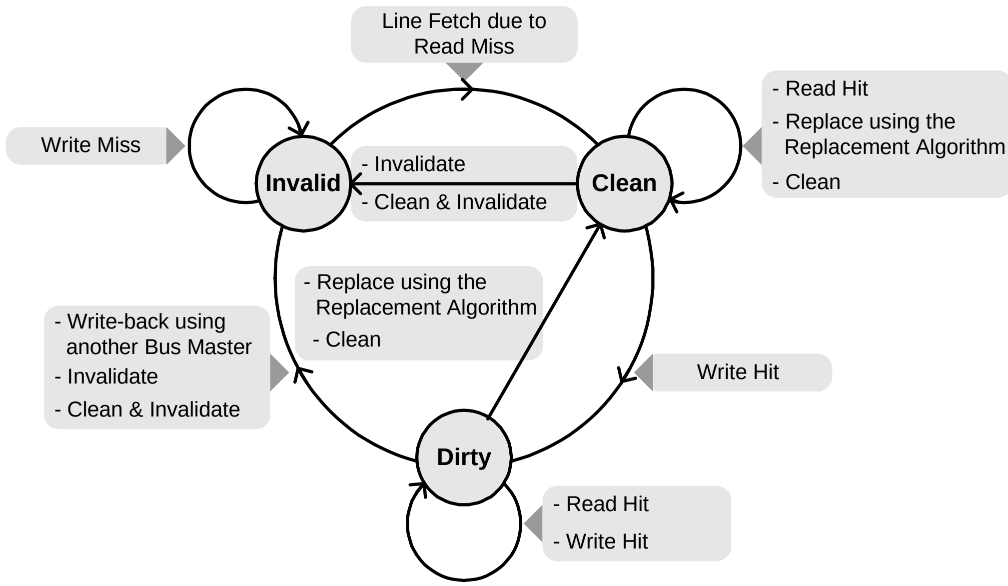
Operations that Must be Managed by the User

Because the ARM946E-S lacks the bus snoop feature, when cached memory is accessed by a bus master other than ARM9 (such as the subprocessor or DMA), the cache lines must be operated manually.

When data are written to memory by a bus master other than ARM9, invalidate the appropriate cache lines.

Also, when memory is read by a bus master other than ARM9, you should clean the cache line beforehand and perform the Wait for the Write buffer to empty operation.

Figure 4-4 : Cache Line State Transitions (Write-Back Mode)



Example

- If the program in main memory is overwritten by an overlay, etc.

  Invalidate the Instruction cache in the appropriate region.

4.5.2 Write-Through Mode

Table 4-5 shows how the flag states define the cache line state in write-through mode.

Table 4-5 : Cache Line States (Write-Through Mode)

State	Flag State		Description
	Valid	Dirty	
Clean	**1**	*****	Cache line is valid, contents match memory
Invalid	**0**	*****	Cache line is invalid

Operations Managed Automatically by the Cache Controller

As shown in Figure 4-5, read misses/hits and write misses/hits for access from the ARM9 as well as state transitions by the replacement algorithm are performed automatically.

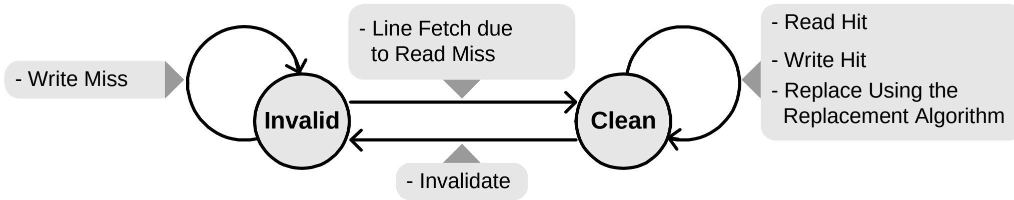
Operations that Must be Managed by the User

Because the ARM946E-S lacks the bus snoop feature, when cached memory is accessed by a bus master other than ARM9 (such as the subprocessor, DMA, etc.), the cache lines must be operated manually.

When data are written to memory by a bus master other than ARM9, you should invalidate the appropriate cache lines.

Also, when memory is read by a bus master other than ARM9, perform the Wait for the Write buffer to empty operation.

Figure 4-5 : Cache Line State Transitions (Write-Through Mode)



Example

- If the program in main memory is overwritten by an overlay, etc.
  Invalidate the Instruction cache in the appropriate region.

5 Display

Note: There are no changes to this chapter other than the addition of the INI bit to the display status.

5.1 Display System

The display system block diagram is shown in "Figure 5-1 : Display System Block Diagram" on page 72.

Each selector in the block diagram can be controlled using the register selection flags in Table 5-1.

Table 5-1 : Selector and Register Selection Flag Map

Selector Name	Register Name	Flag Name
SEL DISP	DISPCNT	Display mode selection
SEL BG0	DISPCNT	2D/3D display selection for BG0
SEL DISP VRAM	DISPCNT	Display VRAM Selection
SEL A	DISPCAPCNT	Capture source A selection
SEL B	DISPCAPCNT	Capture source B selection
SEL CAP	DISPCAPCNT	Capture mode selection
SEL CAP VRAM	DISPCAPCNT	Capture data write destination VRAM selection
SEL LCD	POWCNT	LCD output destination switch

After selecting the graphics display, VRAM display, or main memory display using SEL DISP, the image output becomes Image Output A.

Similarly, the image output of the 2D graphics engine B becomes Image Output B.

Image Outputs A and B each go through Master Brightness Up/Down A and B, respectively, and become the Display Output A and Display Output B that are sent to the LCD.

When finally output to the LCD, these display outputs cannot be layered.

Choose one of the following:

- Send Display Output A to the Upper Screen LCD and send Display Output B to the Lower Screen LCD
- Send Display Output A to the Lower Screen LCD and send Display Output B to the Upper Screen LCD

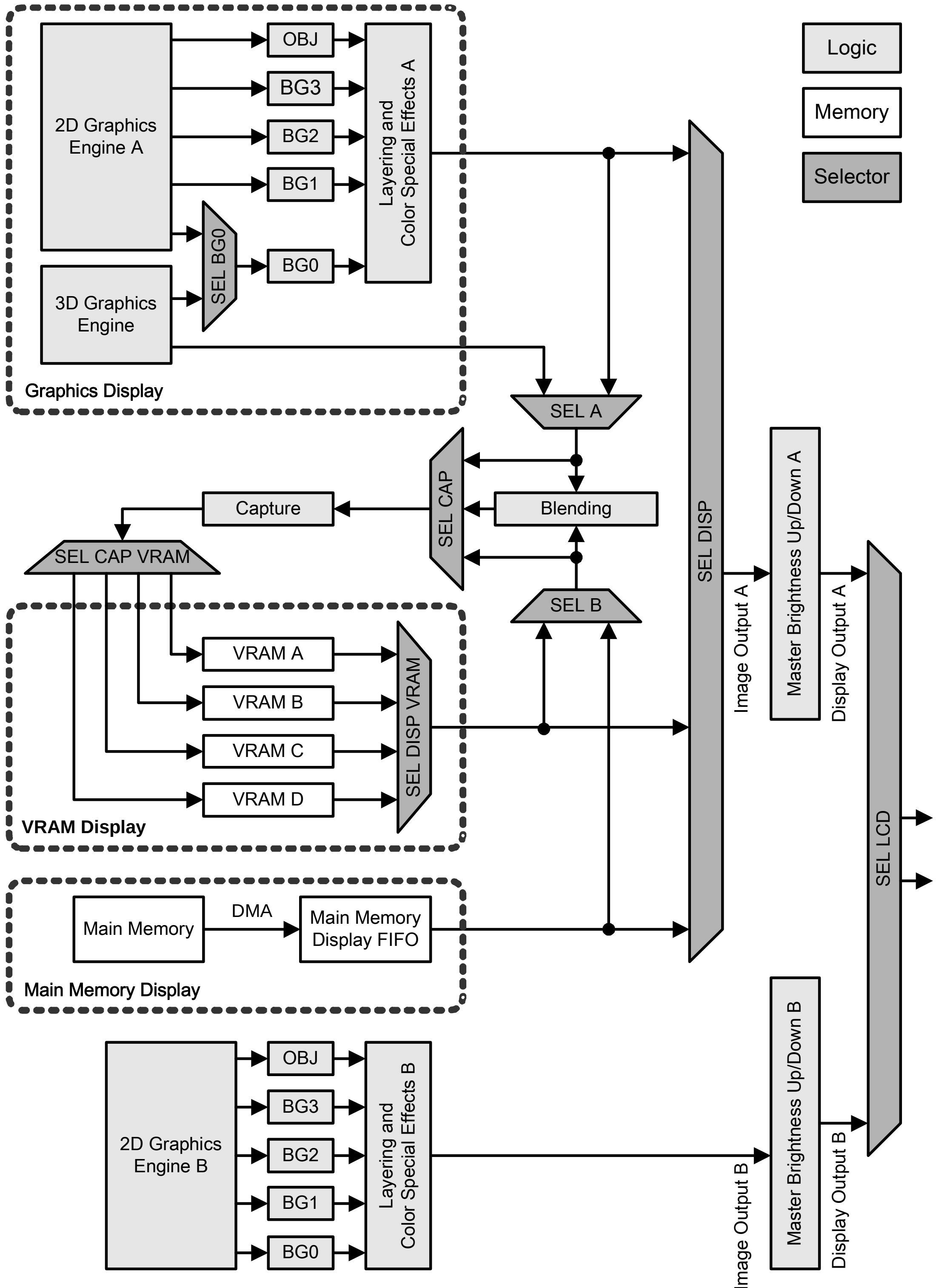
For games that require only one LCD, disable the LCD display you do not use.

For further details, see "11 Power Management" on page 317.

Image Output A allows you to blend and display 2D graphics and 3D graphics in the graphics display.

See "7 3D Graphics" on page 177 for information on the hardware block for 3D image creation.

Figure 5-1 : Display System Block Diagram



5.2 LCD

The specifications for the two LCD controllers included on the TWL are shown below.

5.2.1 LCD Controller Specifications

The LCD clock specifications of the LCD controller are shown in Table 5-2, the LCD scan timing is shown in Figure 5-2, and the specifications for the LCD scan timing are shown in "Table 5-3 : LCD Scan Timing Specifications" on page 74.

Table 5-2 : LCD Clock Specifications

LCD Clock	Frequency (time)
Image Processing Clock	33.513982 Mhz (29.838293 ns)
Dot Clock	5.585664 Mhz (179.029757 ns) (see note)

Note: The Dot Clock is 1/6 of the Image Processing Clock.

Figure 5-2 : LCD Scan Timing

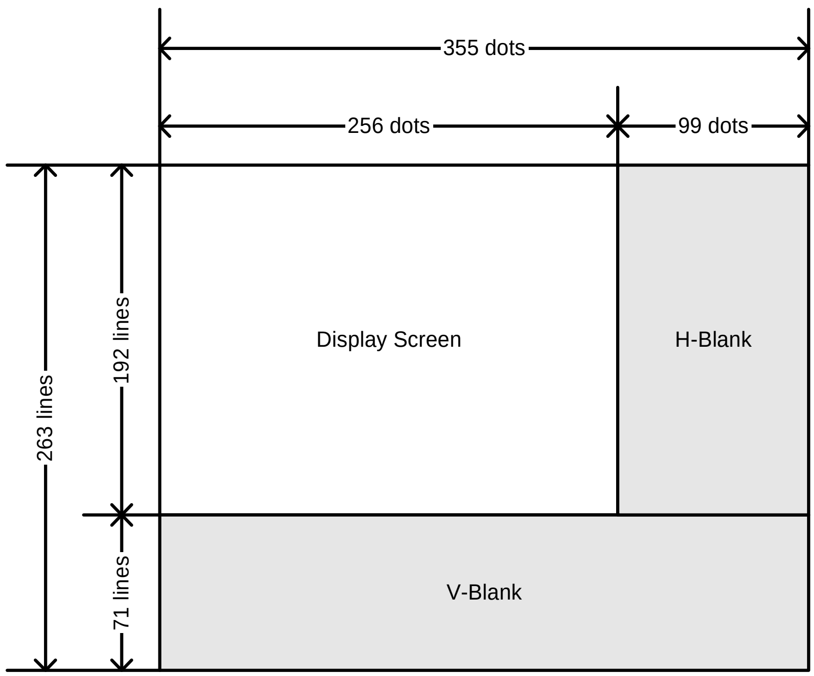


Table 5-3 : LCD Scan Timing Specifications

Item		Spec	Period	Reference: AGB Value
Display Screen Size	Horizontal Dot Count	256 dots	45.8316 µs	240 dots (57.221 µs)
	Vertical Line Count	192 lines	12.2027 ms	160 lines (11.749 ms)
Total Dot Count	Horizontal Dot Count	355 dots	63.5556 µs	308 dots (73.433 µs)
	Vertical Line Count	263 lines	16.7151 ms	228 lines (16.743 ms)
Blanking	H-Blank Dot Count	99 dots	17.7239 µs	68 dots (16.212 µs)
	V-Blank Line Count	71 lines	4.5124 ms	68 lines (4.994 ms)
Scan Cycle	H-Cycle	15.7343 KHz	63.5556 µs	13.618 KHz (73.443 µs)
	V-Cycle	59.8261 Hz	16.7151 ms	59.727 Hz (16.743 ms)

The V-Blank cycle for the 3D rendering engine consists of 23 lines: 191–213. For details, see "7.3 Rendering Engine" on page 251.

5.3 Display Status

DISPSTAT: Display Status Register

Name: DISPSTAT Address: 0x04000004 Attribute: R/W Initial value: 0x0000

15							8	7		5		3	2		0
VC7	VC6	VC5	VC4	VC3	VC2	VC1	VC0	VC8	INI	VQI	HBI	VBI	LYC	HBLK	VBLK
V-Counter Match Setting Values													Status Flags		
									LCD Initialization Signal	Interrupt Request Enable Flags					

- [d15–d07] : V-Counter Match Setting Values

  Note that VC8 is located in d07 (for compatibility with AGB). Values between 0 and 262 can be set. Proper operation is not guaranteed for a value of 263 or higher.

- INI [d06] : LCD Initialization Signal

0	LCD Initialize State
1	LCD Display State

  Writing to this bit is disabled. If the extended features are turned off in the system configuration, the value of this bit will be fixed at 0.

- [d05–d03] : Interrupt Request Enable Flags
  - VQI[d05] : V-Counter match interrupt request enable flag

0	Disable
1	Enable

  - HBI[d04] : H-Blank interrupt request enable flag

0	Disable
1	Enable

    When enabled, H-Blank interrupts are permitted with the Interrupt Enable Register (IE). H-Blank interrupts can be made during the display interval, and also during any of the 263 vertical lines (Line 0 – 262) on the LCD, including V-Blank intervals.

  - VBI[d03] : V-Blank interrupt request enable flag

0	Disable
1	Enable

- [d02–d00] : Status Flag
  - LYC[d02] : V-Counter match detection flag

0	Outside a matching interval
1	During a matching interval

- HBLK[d01] : H-Blank detection flag

0	Outside H-Blank interval
1	During H-Blank interval

- VBLK[d00] : V-Blank detection flag

0	Outside V-Blank interval
1	During V-Blank interval

Note: V-Blank detection flag is set to 1 at the moment it reaches Line 192 and is set to 0 when it reaches Line 262. This is because the OBJ rendering circuitry accesses OBJ-VRAM and OAM starting at Line 262, which is one line before the actual display. In addition, the timing that ends access to OBJ-VRAM and OAM depends on whether the OBJ process is performed during H-Blank. On the other hand, BG-VRAM, BG Palette RAM, and OBJ Palette RAM begin access at Line 0 and end at Line 191. This is summarized in Table 5-4.

Table 5-4 : Period when Graphics Engines Access Memory

<table>
<tr><th colspan="2">Memory Accessed by Graphics Engines</th><th>V-Counter Value</th></tr>
<tr><td rowspan="2">Access period when OBJ rendering circuitry accesses OBJ-VRAM and OAM</td><td>Perform OBJ process during H-Blank</td><td>0-191, 262</td></tr>
<tr><td>Does not perform OBJ process during H-Blank</td><td>0-190, 262</td></tr>
<tr><td colspan="2">Access period when rendering circuitry accesses BG-VRAM, BG Palette RAM, and OBJ Palette RAM</td><td>0-191</td></tr>
<tr><td colspan="2">(Reference) Period when the V-Blank deflection flag becomes 1</td><td>192-261</td></tr>
</table>

VCOUNT: V-Counter Register

Name: VCOUNT Address: 0x04000006 Attribute: R/W Initial value: 0x0000

15							8	7							0
							V8	V7	V6	V5	V4	V3	V2	V1	V0
							V-Counter Values								

- [d08–d00] : V-Counter Values

  Unlike the bit arrangement of the V-Counter Match Setting Values in the DISPSTAT register, these bits are in a normal arrangement.

  1. When reading values

  Can read which of the LCD's total 263 lines is currently displayed. The readout value is between 0 and 262.

  If the readout value is between 0 and 191, images are being drawn. If the value is between 192 and 262, it is a V-Blank period.

  To learn about the LCD's display timing, see "5.2 LCD" on page 73.

  2. When writing values

  Written values are reflected when the hardware's V-Counter is updated.

  By using this register, you can synchronize all TWL V cycles by adjusting the V-count value when communicating among multiple TWL devices.

  Confirm that the current value of the V-Counter is between 202 and 212 and write values only between 202 and 212. Proper operation of the 3D engine is not guaranteed when writing values outside this range.

Note: When there is a conflict between the access to the display circuit VRAM and the access to VRAM from the CPU, the display circuit VRAM takes precedence.

Because the dot clock of the LCD controller is 1/6 of a cycle of the image processing clock and the system clock, the timing for the LCD controller to access the VRAM is once every six cycles.

With this timing, when simultaneously accessing from the CPU, the CPU access must wait for one cycle.

5.4 Display Control

5.4.1 Top LCD/Bottom LCD Output Switching

POWCNT: Power Control Register

Name: POWCNT Address: 0x04000304 Attributes: R/W Initial Value: 0x0000

15						9	8	7				3	2	1	0
DSEL						2DGB						GE	RE	2DGA	MDE

LCD Output Destination Switching Flag

- DSEL[d15] : LCD Output Switching Flag

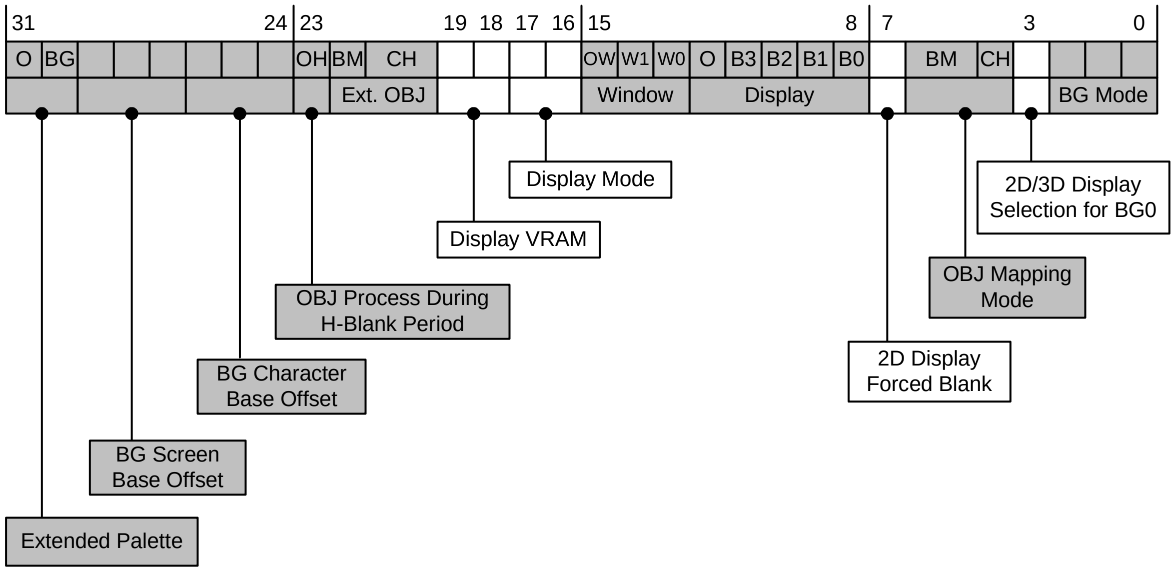
0	Send Display Output A to the Lower Screen LCD Send Display Output B to the Upper Screen LCD
1	Send Display Output A to the Upper Screen LCD Send Display Output B to the Lower Screen LCD

You can switch the LCD output destination with no delay by configuring the Power Control Register.

5.4.2 Display Control of 2D Graphics Engine A

DISPCNT: Display Control Register

Name: DISPCNT Address: 0x04000000 Attribute: R/W Initial value: 0x00000000



- [d19–d18] : Display VRAM

  Selects the VRAM block to display when in VRAM Display Mode (see "[d17–d16] : Display mode").

00	VRAM-A
01	VRAM-B
10	VRAM-C
11	VRAM-D

- [d17–d16] : Display mode

  When the Display mode is OFF, 2D/3D graphics, VRAM display, and main memory display are not selected and appear white.

  Graphics display mode displays both 2D and 3D graphics.

  VRAM display mode displays the bitmap data stored in VRAM.

  Main memory display mode displays the bitmap data stored in main memory (requires a DMA setting).

  For details, see the appropriate sections.

0	Display OFF
1	Graphics Display
2	VRAM Display
3	Main Memory Display

- [d07] : 2D Display Forced Blank

  The 2D graphics display is forcibly halted by the CPU. Because 2D display is halted, 3D graphics using BG0 are not displayed either.

  During a forced blank, the 2D graphics circuitry does not access VRAM, and the LCD screen is white.

  However, even during a forced blank, the internal HV synchronization counter continues to run.

  If the forced-blank setting is changed from ON to OFF during a display period of the internal HV synchronization counter, the effect takes place immediately; if it is changed from OFF to ON, the switch takes place at the start after three lines.

- [d03] : 2D/3D Display Selection for BG0

  This bit determines whether to use one of the BG screens (BG0) for 2D graphics or for 3D graphics.

  When 3D graphics are selected, the 2D graphics features for BG0 are limited and the specifications for color special effects change. See "7.4 2D Graphics Features You Can Apply to the 3D Screen After Rendering" on page 293.

0	Display 2D graphics
1	Display 3D graphics

- Other bits

  The bits in the DISPCNT register not covered above are explained in the following sections:

  Bits related to the display control of 2D graphics features are explained in "6 2D Graphics" on page 97.

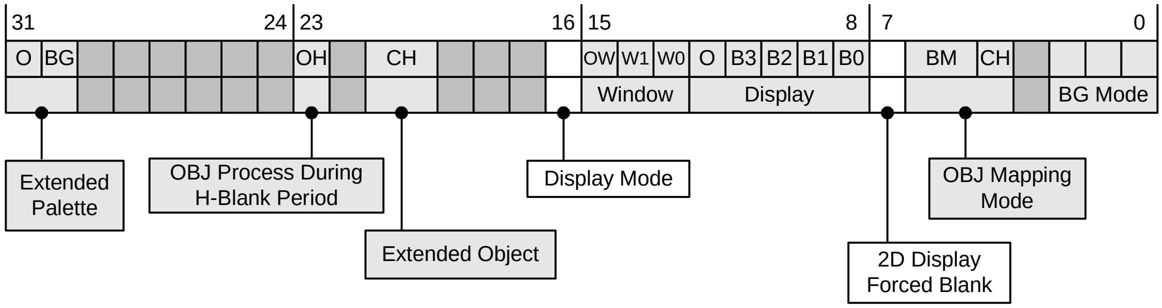
  Bits related to BG are explained in "6.2 BG" on page 101.

  Bits related to OBJ are explained in "6.3 OBJ" on page 133.

5.4.3 2D Graphics Engine B Display Controls

DB_DISPCNT: Display Control Register 1

Name: DB_DISPCNT Address: 0x04001000 Attributes: R/W Initial Value: 0x00000000



- [d16] : Display Mode

0	Display OFF
1	Display ON

- [d07] : 2D Display Forced Blank

  This forcibly stops the 2D graphics engine circuit using the CPU.

  During a forced blank, the 2D graphics circuit does not access the VRAM, and the LCD screen appears white.

  However, the internal HV synchronization counter runs even during a forced blank.

  If the internal HV counter changes a forced blank during the display interval, the ON/OFF switches immediately after configuration when going from ON to OFF or switches from the top after three lines when going from OFF to ON.

5.4.4 Display Modes

As indicated in Table 5-5, on the Display Output A side (the 2D Graphics Engine A), there are modes that display the bitmap data in the VRAM and main memory in addition to the mode that displays the images generated by the graphics circuit.

Table 5-5 : Overview of the Display Modes (2D Graphics Engine A)

Display Mode Number	Display Mode	Display Size	Frame Rate	Features			
				3D Display	Character BG Display	Bitmap BG Display	OBJ Display
0	Display OFF	-	-	-	-	-	-
1	Graphics Display	256x192	60 fps	X	X	X	X
2	VRAM Display	256x192	60 fps			X	
3	Main Memory Display	256x192	60 fps			X	

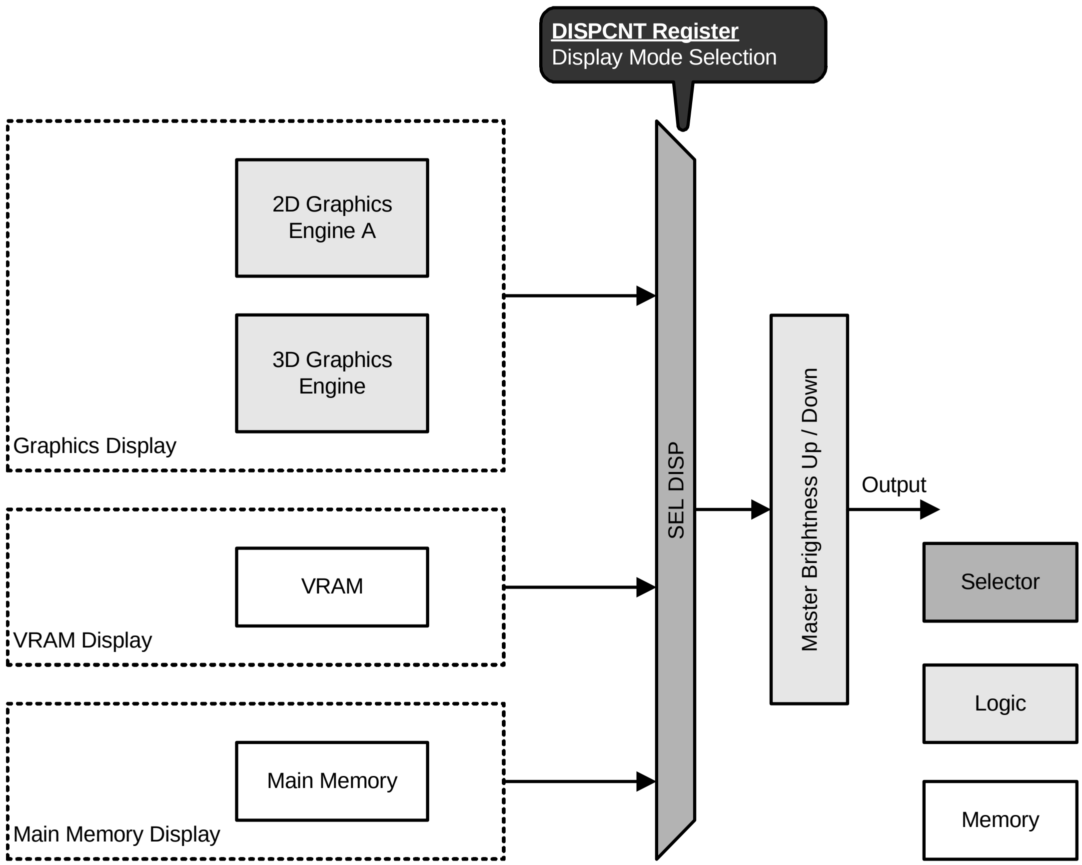
As indicated in Table 5-6, on the Display Output B side (the 2D graphics engine B), the only mode selection is graphics display ON or OFF.

Table 5-6 : Overview of the Display Modes (2D Graphics Engine B)

Display Mode Number	Display Mode	Display Size	Frame Rate	Feature		
				Character BG Display	Bitmap BG Display	OBJ Display
0	Display OFF	-	-	-	-	-
1	Graphics Display	256x192	60 fps	X	X	X

"Figure 5-3 : Display Mode Selection (Display Output A Side Only)" on page 83 is a simplified version of the display mode for Display Output A in "Figure 5-1 : Display System Block Diagram" on page 72.

Figure 5-3 : Display Mode Selection (Display Output A Side Only)



5.4.4.1 Graphics Display Mode

This mode displays images generated with the 2D and 3D graphics features.

See "Figure 5-1 : Display System Block Diagram" on page 72 for information about the entire display system.

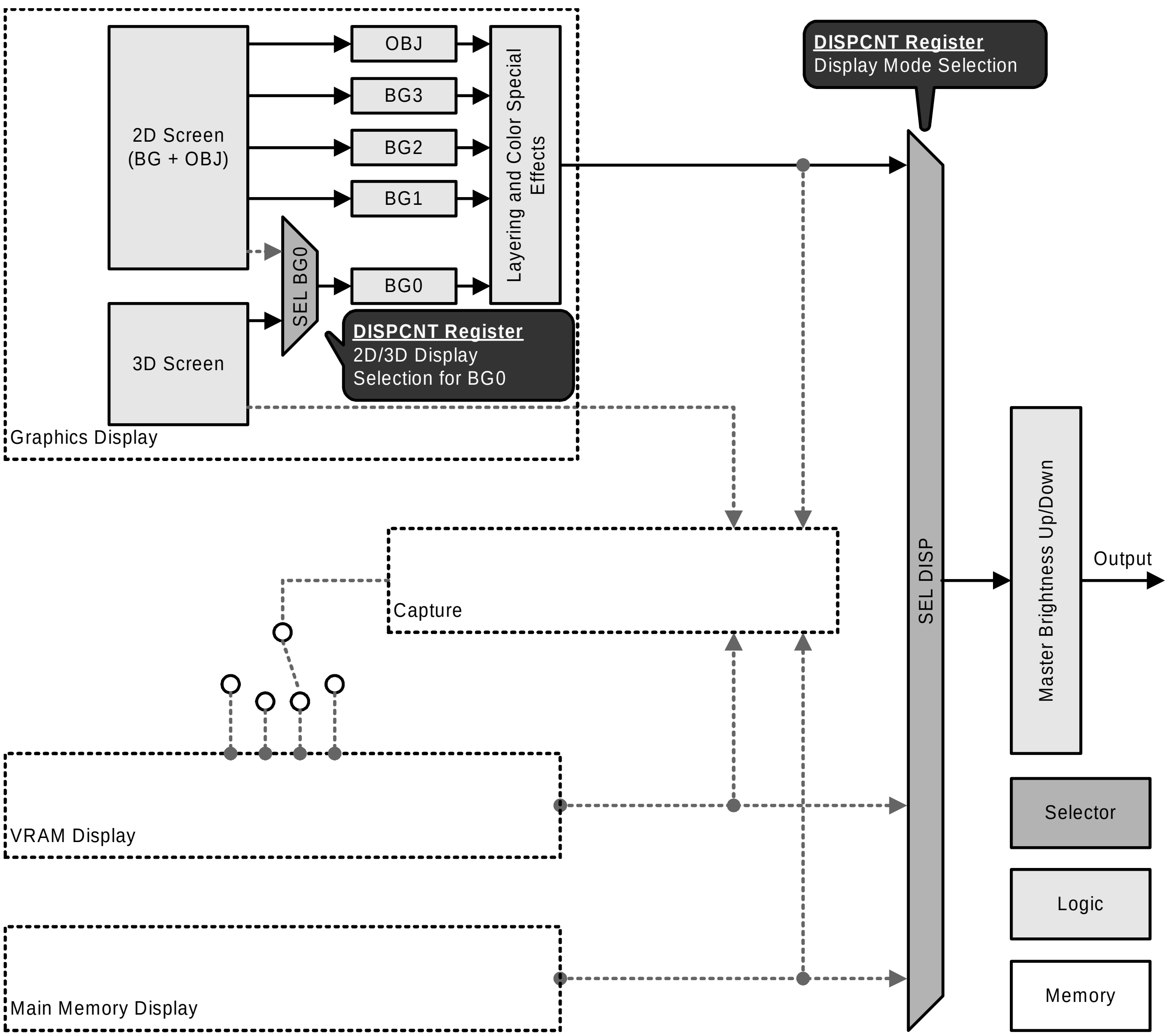
To read about the various graphic features, see "6 2D Graphics" on page 97 and "7 3D Graphics" on page 177.

- **Graphics display mode – Example 1**

The example in Figure 5-4 shows how the results of 3D rendering are layered with the 2D screen and displayed.

Although 3D display is handled as the BG0 screen, the 2D graphics features of BG0 are limited, and the color special effect specifications have changed. See "7.4 2D Graphics Features You Can Apply to the 3D Screen After Rendering" on page 293.

Figure 5-4 : Display Mode Selection (Display Output A Side Only)



- **Graphics display mode – Example 2**

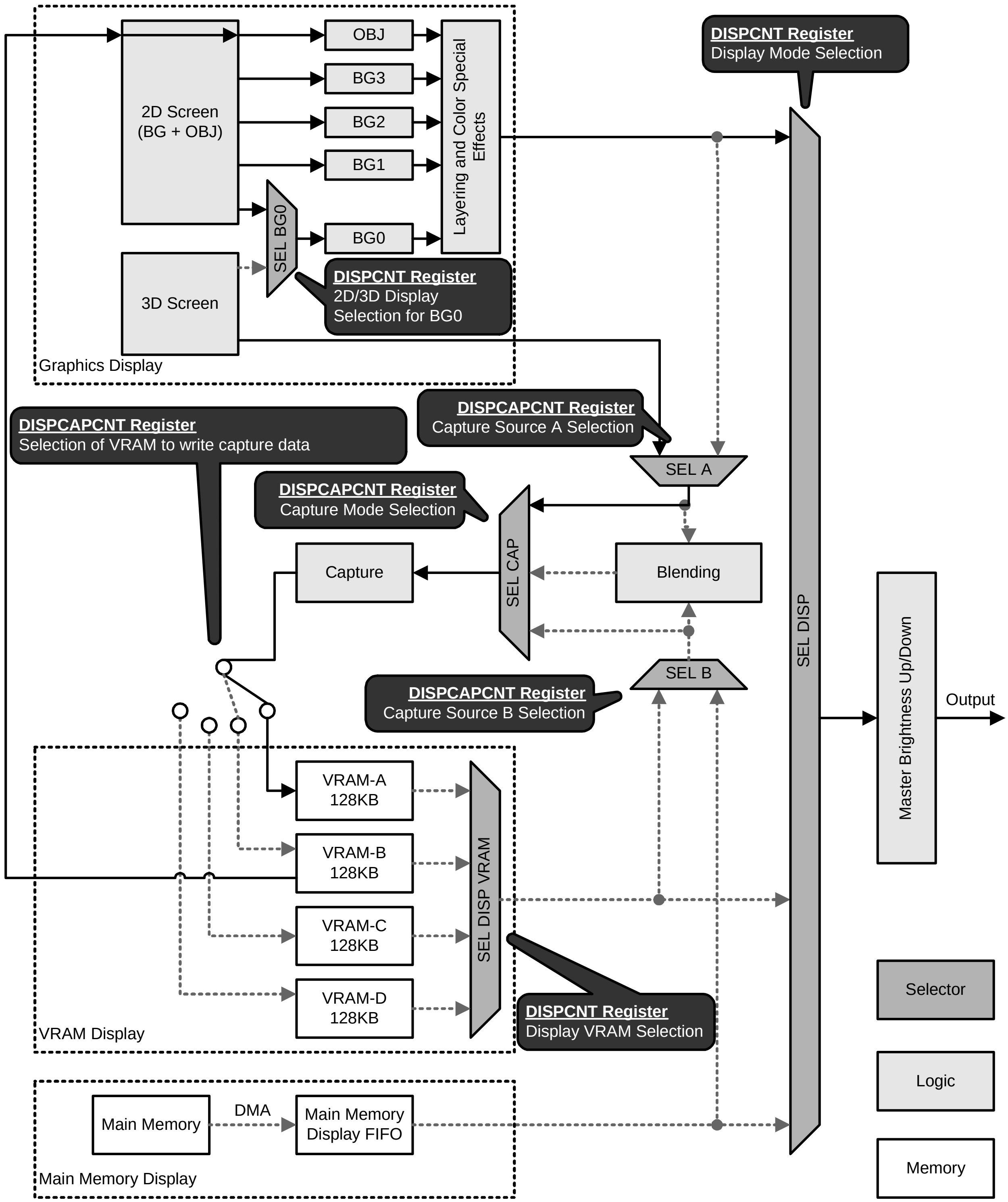
  In the example shown in "Figure 5-5 : Example of Displaying the Bitmap OBJ Results of 3D Rendering" on page 86, the results of 3D rendering are pasted in a bitmap OBJ and displayed.

  The rendering engine's clear alpha value is set to 0, and the 3D rendering result is captured. Then, in the next frame, the VRAM is assigned to a bitmap OBJ, according to the value of the RAM Bank Control register. This enables the 3D rendering result to be displayed as an OBJ.

  At this moment in the sequence, alpha value segments that remain 0 in the 3D alpha-blending process are transparent. (See "7.3.7 Alpha-Blending" on page 283 for the capture feature and the rendering engine.)

  In this example, double buffering occurs by alternately assigning VRAM-A and VRAM-B to LCDC and OBJ-VRAM.

Figure 5-5 : Example of Displaying the Bitmap OBJ Results of 3D Rendering



5.4.4.2 VRAM Display Mode

When the DISPCNT register is set for VRAM display mode, one frame of bitmap data stored in a VRAM block is shown from the start of the next display. The DISPCNT register can specify which VRAM block to use.

VRAM is displayed using a different system than the 2D circuitry and the 3D circuitry, so when the mode is set to VRAM display mode, images can be created by the graphics circuitry and captured to VRAM at the same time that images are being displayed. (See "Figure 5-1 : Display System Block Diagram" on page 72).

You can specify the same VRAM block for display and for capturing images.

For details about capturing images, see "5.5 Display Capture" on page 91.

The pixel data format for VRAM display mode is shown below.

VRAM Display Mode Data Format

15	14 10	9 8 7 5	4 0
	BLUE	GREEN	RED
	Pixel Color Data		

Figure 5-6 shows the VRAM address map of the LCD pixels.

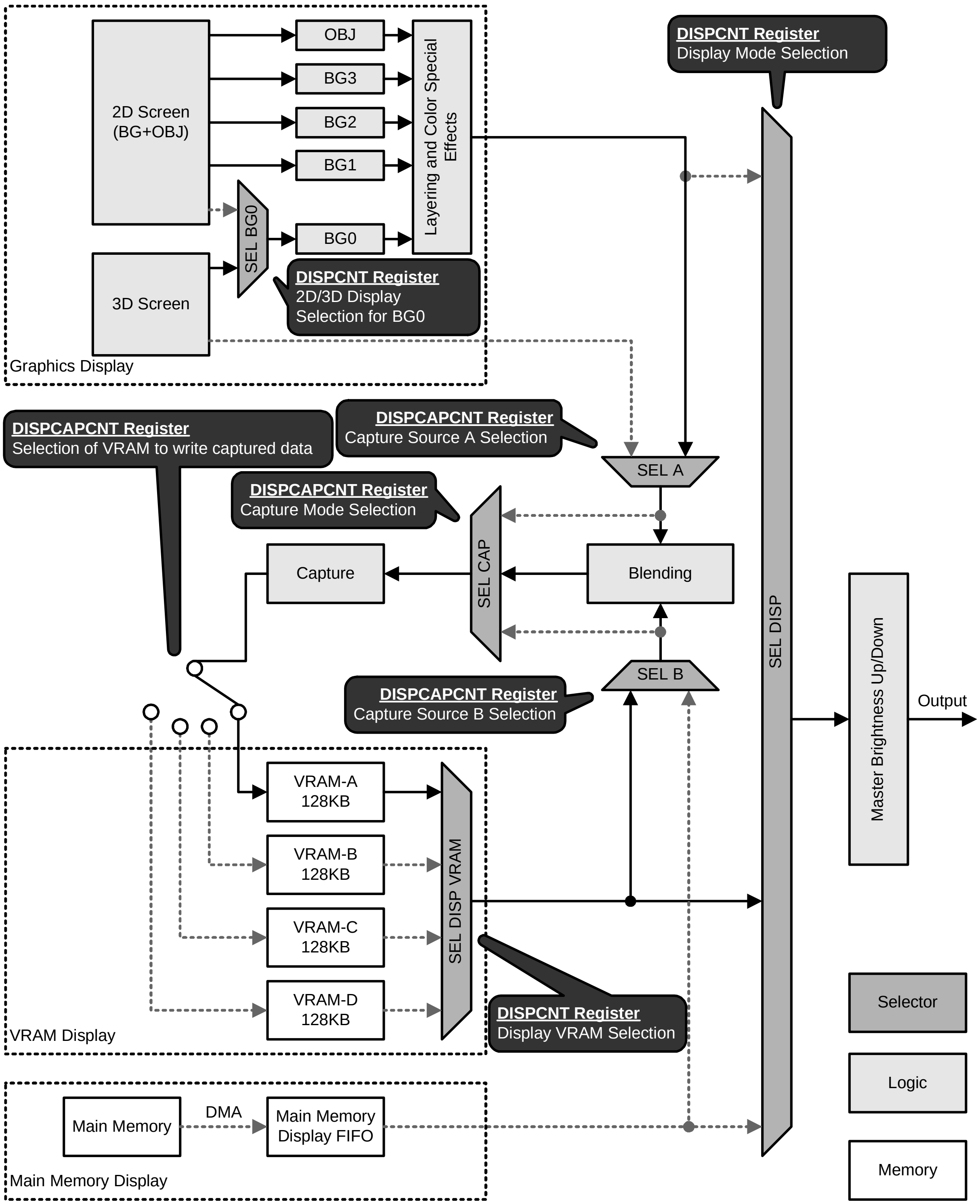
Figure 5-6 : VRAM Address Map of the LCD Pixels

	Dot 0	1	2	3		253	254	255
Line 0	0h	2h	4h	6h		1FAh	1FCh	1FEh
1	200h	202h	204h	206h		3FAh	3FCh	3FEh
2	400h	402h					5FCh	5FEh
3	600h	602h					7FCh	7FEh
4	800h							9FEh
187	17600h							177FEh
188	17800h	17802h					179FCh	179FEh
189	17A00h	17A02h					17BFCh	17BFEh
190	17C00h	17C02h	17C04h	17C06h		17DFAh	17DFCh	17DFEh
191	17E00h	17E02h	17E04h	17E06h		17FFAh	17FFCh	17FFEh

- VRAM display mode - Example

  In "Figure 5-7 : Example of the Motion Blur Effect that Uses the Display Capture" on page 88, the image created by the graphic circuitry is put into VRAM using the capture feature, and then the image is displayed with the mode set to VRAM Display. When the image is captured, a motion blur effect is achieved by blending with the display-use VRAM.

Figure 5-7 : Example of the Motion Blur Effect that Uses the Display Capture



5.4.4.3 Main Memory Display Mode

This mode enables the display of bitmap data held in main memory. When the DISPCNT register is set to main memory display mode, the data held in the main memory display FIFO is transferred to the LCD module at the beginning of the next display. A data request is sent to DMA for every data transfer.

There is a four-layer FIFO between the main memory display FIFO register and the LCD module, and the LCD module takes four words at a time. For this reason, you should write four layers of data at a time to the main memory display FIFO register.

To be more specific, after setting the DMA transfer bit width to 32 bits and the word count to 4, set the DMA startup mode to main memory display mode. For this mode, be sure to set the DMA source address to the main memory region.

Table 5-7 shows the DMA configuration when using the main memory display mode.

Table 5-7 : DMA Configuration when Using the Main Memory Display Mode

Setting	Value
Source Address	Main memory
Transfer Bit Width	32 bits
Word Count	4

For details about the DMA configuration, see "8 DMA" on page 297.

Data from main memory is displayed using a system other than the 2D and 3D circuitry, so when the mode is set to main memory display, images can be created by the graphics circuitry and captured to VRAM at the same time that images are being displayed. (See "Figure 5-1 : Display System Block Diagram" on page 72.)

Main Memory Display FIFO Register

Name: DISP_MMEM_FIFO Address: 0x04000068 Attribute: R/W Initial value: 0x00000000

31	30–26	25–21	20–16	15	14–10	9–5	4–0
	BLUE	GREEN	RED		BLUE	GREEN	RED
ODD				EVEN			

"Figure 5-8 : LCD Pixel EVEN/ODD Map of the Main Memory Display FIFO Register" on page 90 shows the LCD pixel EVEN/ODD map of the main memory display FIFO register.

Figure 5-8 : LCD Pixel EVEN/ODD Map of the Main Memory Display FIFO Register

	Dot 0	1	2	3		253	254	255
Line 0	EVEN	ODD	EVEN	ODD		ODD	EVEN	ODD
1	EVEN	ODD	EVEN	ODD		ODD	EVEN	ODD
2	EVEN	ODD					EVEN	ODD
3	EVEN	ODD					EVEN	ODD
4	EVEN							ODD
187	EVEN							ODD
188	EVEN	ODD					EVEN	ODD
189	EVEN	ODD					EVEN	ODD
190	EVEN	ODD	EVEN	ODD		ODD	EVEN	ODD
191	EVEN	ODD	EVEN	ODD		ODD	EVEN	ODD

5.5 Display Capture

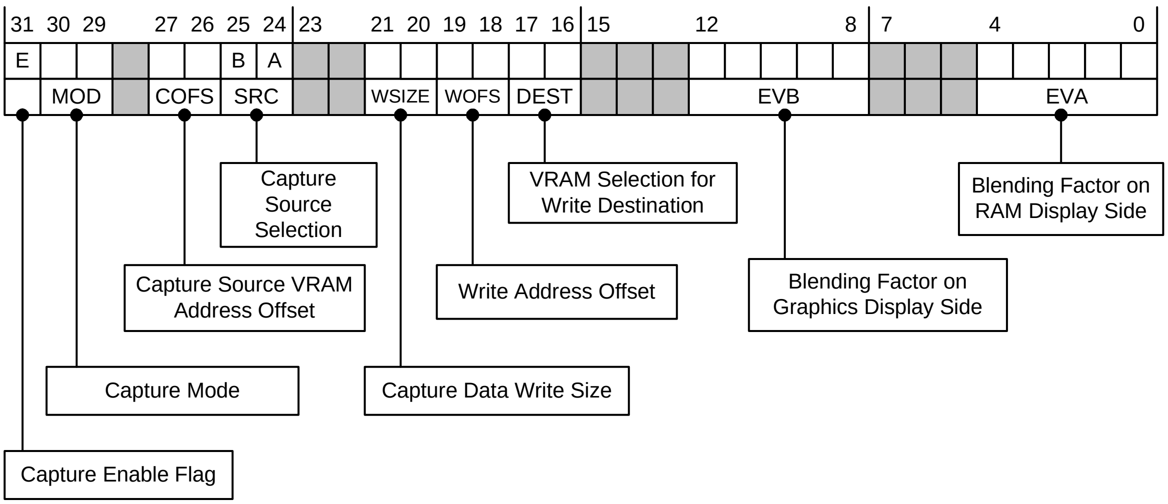
This feature enables 2D and 3D graphics and image output from VRAM and main memory to be read into VRAM.

Only the image output on the Display Output A side (the 2D graphics engine) can be captured.

It also enables images from two sources to be blended, and then captured.

DISPCAPCNT: Display Capture Control Register

Name: DISPCAPCNT Address: 0x04000064 Attribute: R/W Initial value: 0x00000000



- E[d31] : Display Capture Enable Flag

  When the flag is set to 1, one screen of data is captured from the next 0 line, and then the flag is set to 0.

0	Disable
1	Enable

- MOD[d30–d29] : Capture Mode

00	Capture data from source A
01	Capture data from source B
10	Capture the result of blending data from sources A and B
11	

- COFS[d27–d26] : Read Address Offset for Capture Data Source VRAM

  Invalid in VRAM display mode.

  If the offset exceeds 0x20000 during reading, the reading continues after wrapping to address 0x00000.

00	0x00000
01	0x08000
10	0x10000
11	0x18000

- SRC[d25–d24] : Capture Data Source Selection

  B

0	VRAM
1	Main Memory

  A

0	Graphics display screen (after 3D/2D blending)
1	3D screen

- WSIZE[d21–d20] : Capture Size

  Specifies the size when writing the capture data. With RAM captures, one line is always read as a 256-dot image, so you cannot blend and then capture (see Capture mode above) when the setting is 128x128 dots.

00	128x128 dots (0x08000 bytes)
01	256x64 dots (0x08000 bytes)
10	256x128 dots (0x10000 bytes)
11	256x192 dots (0x18000 bytes)

- WOFS[d19–d18] : Address Offset for Capture Data Write

  This can specify the offset value for the address where data is written in the specified VRAM. If the offset exceeds 0x20000 during writing, the writing continues after wrapping to address 0x00000.

00	0x00000
01	0x08000
10	0x10000
11	0x18000

- DEST[d17–d16] : Capture Data Write Destination VRAM Selection

  The write destination VRAM must be allocated to the LCDC.

00	VRAM-A
01	VRAM-B
10	VRAM-C
11	VRAM-D

- EVB[d12–d08], EVA[d04–d00] : Blending Factors

  Sets the blending factors for capture sources A and B. See below for the calculation method.

  In VRAM display mode, you can set the same VRAM block for display VRAM and for writing the captured image data.

Capture Data Format

15	14–10	9–5	4–0
A	BLUE	GREEN	RED
α	Pixel Color Data		

Although 3D graphics are output in R:G:B=6:6:6 color, because capture occurs in R:G:B=5:5:5 color (employing the upper 5 bits), the image gradient becomes a little coarse.

Figure 5-9 shows the LCD pixel map of the capture data when the capture size is 256 x 192 dots.

Figure 5-9 : LCD Pixel Map of the Capture Data (When the Capture Size is 256 x 192 Dots)

	Dot 0	1	2	3		253	254	255
Line 0	0h	2h	4h	6h		1FAh	1FCh	1FEh
1	200h	202h	204h	206h		3FAh	3FCh	3FEh
2	400h	402h					5FCh	5FEh
3	600h	602h					7FCh	7FEh
4	800h							9FEh
187	17600h							177FEh
188	17800h	17802h					179FCh	179FEh
189	17A00h	17A02h					17BFCh	17BFEh
190	17C00h	17C02h	17C04h	17C06h		17DFAh	17DFCh	17DFEh
191	17E00h	17E02h	17E04h	17E06h		17FFAh	17FFCh	17FFEh

TWL-06-0017-001-D
Released: February 16, 2009

- **How to calculate data to write**

1. For data captured from source A:

   CAP = Ca

   Capture source A's alpha value is used for the alpha value.

2. For data captured from source B:

   CAP = Cb

   Capture source B's alpha value is used for the alpha value

3. For capturing data blended from sources A and B:

$$CAP = \frac{(Ca \times Aa \times EVA) + (Cb \times Ab \times EVB)}{16}$$

The alpha value is 1 when EVA is non-zero and capture source A's alpha value is 1, or when EVB is non-zero and capture source B's alpha value is 1. In all other circumstances, the alpha value is 0.

CAP: The color to write (calculation results are rounded to the nearest integer)

Ca: A's capture source data color, EVA: Blending factor for A

Cb: B's capture source data color, EVB: Blending factor for B

Aa: A's alpha value: A's capture source alpha value.

Determined as shown below.

Capture Source A Selection	3D Screen Alpha Value	Aa
0	-	1
1	0	0
	1 - 31	1

Ab: alpha value of B: alpha value of B's capture source

Note: When a conflict occurs between access to the display circuit VRAM and access to VRAM from the CPU, the display circuit VRAM access takes precedence.

Because the dot clock of the LCD controller is 1/6 of a cycle of the image processing clock and the system clock, the timing for the LCD controller to access the VRAM is once every six cycles.

If the VRAM of the capture is being displayed while display capturing, the frequency at which the display circuit accesses the VRAM is doubled, and the VRAM is accessed with a timing of once every three cycles.

With this timing, when simultaneously accessing from the CPU, the CPU access must wait one cycle.

5.6 Master Brightness

The brightness up/down process for Image Output A is handled by the configuration of the MASTER_BRIGHT register, while the brightness up/down process for Image Output B is handled by the configuration of the DB_MASTER_BRIGHT register.

MASTER_BRIGHT: Master Brightness Up/Down Register

Name: MASTER_BRIGHT Address: 0x0400006C Attribute: R/W Initial value: 0x0000

15 14										4 0
E_MOD										E_VALUE
Mode										Factor

DB_MASTER_BRIGHT: Master Brightness Up/Down B Register

Name: DB_MASTER_BRIGHT Address: 0x0400106C Attribute: R/W Initial value: 0x0000

15 14										4 0
E_MOD										E_VALUE
Mode										Factor

The MASTER_BRIGHT and DB_MASTER_BRIGHT registers share identical configuration details.

- E_MOD [d15–d14] : Mode

  Sets the mode for processing brightness up/down.

Setting	Process
00	No change in brightness
01	Increase brightness
10	Decrease brightness
11	Setting prohibited

- E_VALUE [d04–d00] : Factor

  Sets the factors as calculated below.

  1. Brightness up computation

  Rout = Rin + (63 – Rin) x (E_VALUE/16)

  Gout = Gin + (63 – Gin) x (E_VALUE/16)

  Bout = Bin + (63 – Bin) x (E_VALUE/16)

  2. Brightness down computation

  Rout = Rin – Rin x (E_VALUE/16)

  Gout = Gin – Gin x (E_VALUE/16)

  Bout = Bin – Bin x (E_VALUE/16)

  The result of the Brightness Up and Down computation (Rout, Gout, and Bout) is rounded to the nearest integer.

6 2D Graphics

TWL's 2D graphics engines are unchanged from NITRO except for circuit revisions (bug fixes) to the BG window features. For details on the changes, refer to "6.6 Windows" on page 166.

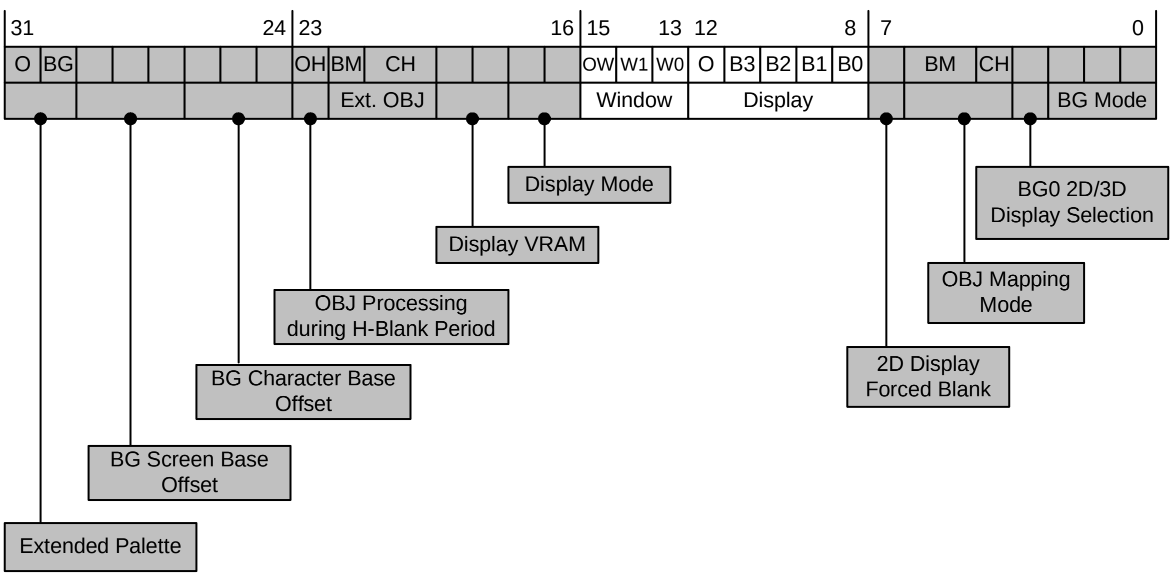
TWL has two 2D graphics engines: 2D Graphics Engine A and 2D Graphics Engine B. 2D Graphics Engine A can use 3D Graphics BG and large-screen 256-color Bitmap BG, but 2D Graphics Engine B cannot. In the following sections, 2D Graphics Engine A is sometimes referred to as 2D_A and 2D Graphics Engine B as 2D_B. Where register names differ for 2D Graphics Engine A and 2D Graphics Engine B, the register name for 2D Graphics Engine B is given inside square brackets [ ].

6.1 Controlling the 2D Display

The display for each 2D graphics feature can be controlled and turned on or off independently. Control register settings differ for 2D Graphics Engine A and 2D Graphics Engine B.

DISPCNT: Display Control Register (2D Graphics Engine A)

Name: DISPCNT Address: 0x04000000 Attribute: R/W Initial Value: 0x00000000



- [d15–d13] : Window Display Enable Flag

  See "6.6 Windows" on page 166 to read about the window features.

  - OW [d15] : OBJ Window Display Enable Flag

0	Disable display
1	Enable display

  To display the OBJ Window requires enabling both the OBJ Window Display Enable Flag and the OBJ Display Enable Flag.

  - W1 [d14] : Window 1 Display Enable Flag

0	Disable display
1	Enable display

- W0 [d13] : Window 0 Display Enable Flag

0	Disable display
1	Enable display

- [d12–d08] : Display Selection Flag
  - O [d12] : OBJ Display Enable Flag

0	Disable display
1	Enable display

  - B3 [d11] : BG3 Display Enable Flag

0	Disable display
1	Enable display

  - B2 [d10] : BG2 Display Enable Flag

0	Disable display
1	Enable display

  - B1 [d09] : BG1 Display Enable Flag

0	Disable display
1	Enable display

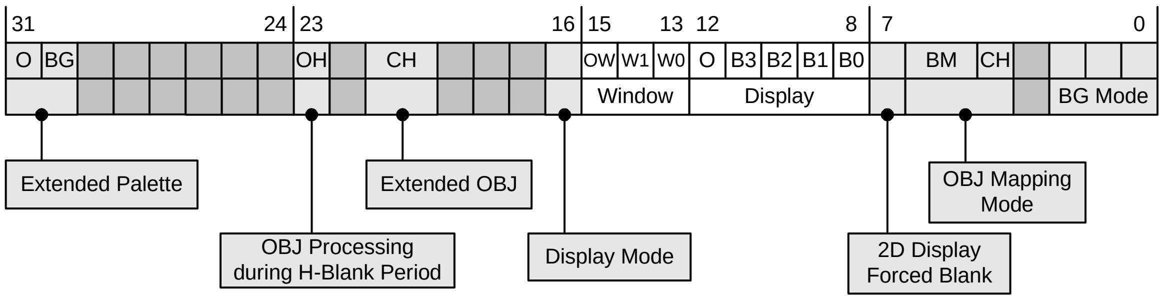
  - B0 [d08] : BG0 Display Enable Flag

0	Disable display
1	Enable display

Note: The [d12–d08] Display Selection Flags are applied at different times depending on whether display is enabled or disabled. When one of these flags is toggled from 1 to 0, display is disabled immediately. When one of these flags is toggled from 0 to 1, display is enabled beginning from the start of the line three lines later.

DB_DISPCNT: Display Control Register 1 (2D Graphics Engine B)

Name: DB_DISPCNT Address: 0x04001000 Attribute: R/W Initial Value: 0x00000000



- [d15–d13] : Window Display Enable Flag

  See "6.6 Windows" on page 166 to read about the window features.

  - OW [d15] : OBJ Window Display Enable Flag

0	Disable display
1	Enable display

To display the OBJ Window requires enabling both the OBJ Window Display Enable Flag and the OBJ Display Enable Flag.

  - W1 [d14] : Window 1 Display Enable Flag

0	Disable display
1	Enable display

  - W0 [d13] : Window 0 Display Enable Flag

0	Disable display
1	Enable display

- [d12–d08] : Display Selection Flag
  - O [d12] : OBJ Display Enable Flag

0	Disable display
1	Enable display

  - B3 [d11] : BG3 Display Enable Flag

0	Disable display
1	Enable display

  - B2 [d10] : BG2 Display Enable Flag

0	Disable display
1	Enable display

- B1 [d09] : BG1 Display Enable Flag

0	Disable display
1	Enable display

- B0 [d08] : BG0 Display Enable Flag

0	Disable display
1	Enable display

Note: The [d12–d08] Display Selection Flags are applied at different times depending on whether display is enabled or disabled. When one of these flags is toggled from 1 to 0, display is disabled immediately. When one of these flags is toggled from 0 to 1, display is enabled beginning from the start of the line three lines later.

6.2 BG

6.2.1 BG Mode

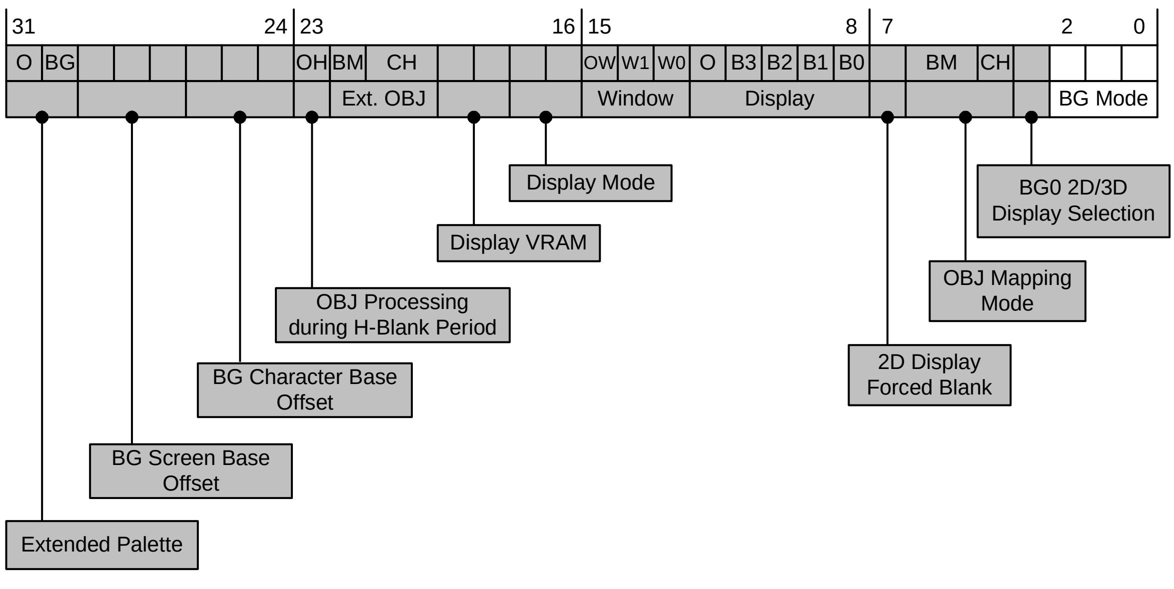
The BG modes that can be set for 2D Graphics Engine A and 2D Graphics Engine B are different.

6.2.1.1 2D Graphics Engine A

With 2D Graphics Engine A, BG0 can be displayed as either 2D or 3D. In addition, Large-Screen 256-Color Bitmap BG can be selected as the BG type for BG2.

DISPCNT: Display Control Register (2D Graphics Engine A)

Name: DISPCNT Address: 0x04000000 Attribute: R/W Initial Value: 0x00000000



- [d02–d00] : BG Mode

  These bits set the BG mode number. The BG mode selects the BG types that can be used. See Table 6-1 for a list of BG modes for 2D Graphics Engine A.

  The DISPCNT register can be used to select either Text BG or 3D BG as the BG type for BG0. See "7 3D Graphics" on page 177 for details on 3D BG display.

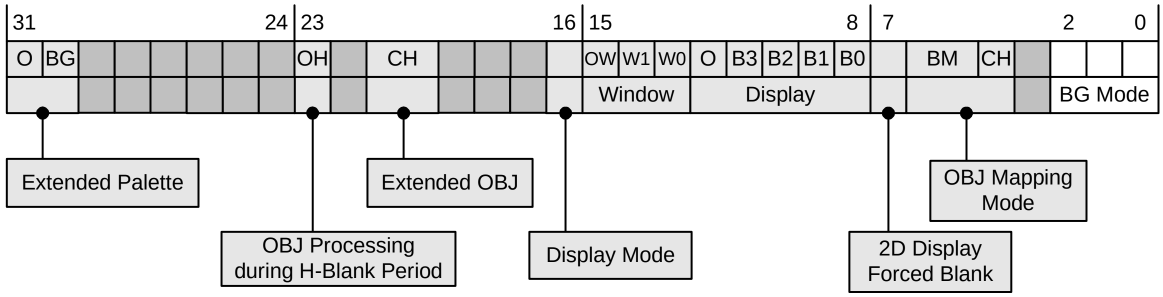
Table 6-1: List of BG Modes (2D Graphics Engine A)

BG Mode Number	BG0	BG1	BG2	BG3
0	Text BG/3D BG	Text BG	Text BG	Text BG
1	Text BG/3D BG	Text BG	Text BG	Affine BG
2	Text BG/3D BG	Text BG	Affine BG	Affine BG
3	Text BG/3D BG	Text BG	Text BG	Affine Extended BG
4	Text BG/3D BG	Text BG	Affine BG	Affine Extended BG
5	Text BG/3D BG	Text BG	Affine Extended BG	Affine Extended BG
6	3D BG	—	Large-Screen 256-Color Bitmap BG	—
7	Prohibited Setting			

6.2.1.2 2D Graphics Engine B

DB_DISPCNT: Display Control Register 1 (2D Graphics Engine B)

Name: DB_DISPCNT Address: 0x04001000 Attribute: R/W Initial Value: 0x00000000



- [d02–d00] : BG Mode

  These bits set the BG mode number. The BG mode selects the BG types that can be used. See Table 6-2 for a list of BG modes for 2D Graphics Engine B.

  Note: Unlike 2D Graphics Engine A, Large-screen 256-Color Bitmap BG cannot be set as the BG type for BG2. Furthermore, 3D BG display cannot be set as the BG type for BG0.

Table 6-2 : List of BG Modes (2D Graphics Engine B)

BG Mode Number	BG0	BG1	BG2	BG3
0	Text BG	Text BG	Text BG	Text BG
1	Text BG	Text BG	Text BG	Affine BG
2	Text BG	Text BG	Affine BG	Affine BG
3	Text BG	Text BG	Text BG	Affine Extended BG
4	Text BG	Text BG	Affine BG	Affine Extended BG
5	Text BG	Text BG	Affine Extended BG	Affine Extended BG
6	Prohibited Setting			
7	Prohibited Setting			

6.2.1.3 Basic Features for Each Type of BG

Each BG type has its own special features as described in Table 6-3.

Table 6-3 : Basic Features of BG Types

BG Type	Features
3D BG	This type can display images generated by the 3D graphics engine. It can be displayed with other BG screens according to alpha-blending and priority settings. 2D Graphics Engine B cannot use this type.
Text BG	This type is a character format BG. Text BG is the only BG type that can handle characters defined in 16 colors and control VRAM consumption, but it cannot accommodate affine transformations.
Affine BG	This type is the character format BG that can accommodate affine transformations. It cannot perform character-unit processes (such as HV Flips).
Affine Extended BG	Three types are available: • Character BG that can use 256 colors x 16 palettes • 256-Color Bitmap BG • Direct Color Bitmap BG that can specify color directly
Large-Screen 256-Color Bitmap BG	This type is the Large-Screen Bitmap BG. Because one screen makes full use of the maximum capacity of BG-VRAM (512 KB), it cannot be used together with other BGs. However, it can be used together with a 3D screen. 2D Graphics Engine B cannot use this type.

6.2.1.4 Specifications for Different BG Types

The specifications for the different types of BG are shown in Table 6-4.

Table 6-4 : Specifications for BG Types

Category	Character BG						Bitmap BG
BG Type / Specs	3DBG	Text BG	Affine BG	Affine Extended BG			Large-Screen 256-Color Bitmap BG
				256-Color x 16 Palettes	256-Color	Direct Color	
Screen Size	256x192	256x256 512x256 256x512 512x512	128x128 256x256 512x512 1,024x1,024	128x128 256x256 512x512 1,024x1,024	128x128 256x256 512x256 512x512	128x128 256x256 512x256 512x512	512x1,024 1,024x512
Specifiable Character Count	—	1,024	256	1,024	—	—	—
Number of Colors/Palettes	262,144	16/16 256/1 256/16	256/1	256/16	256/1	32,768	256/1
Affine			X	X	X	X	X
HV Flip		X		X			
H Scroll	X	X	X	X	X	X	X
V Scroll		X	X	X	X	X	X
Mosaic		X	X	X	X	X	X
Fade-in/Fade-out	X	X	X	X	X	X	X
Alpha Blending	X	X	X	X	X	X	X
Priority	X	X	X	X	X	X	X

Note 1: 2D Graphics Engine B cannot set 3D BG and Large-Screen 256-Color Bitmap BG as BG types.

Note 2: Because the allocation of BG-VRAM to 2D Graphics Engine B is limited, the following settings cannot be used:

- 256-Color Bitmap: 512x512
- Direct Color Bitmap: 512x256 and 512x512

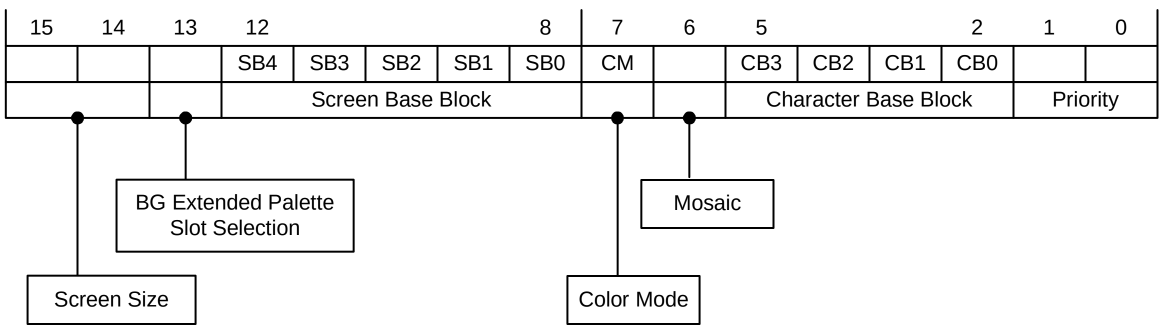
6.2.2 BG Control

There are four BG control registers that correspond to the number of BG screens. With 2D Graphics Engine A, the BG screens are controlled with the BG0CNT, BG1CNT, BG2CNT, and BG3CNT registers.With 2D Graphics Engine B, the BG screens are controlled with the DB_BG0CNT, DB_BG1CNT, DB_BG2CNT, and DB_BG3CNT registers.

Note: 2D Graphics Engine A and 2D Graphics Engine B use different register names as well as different methods to calculate base address values for BG screen data and BG character data.

BGx(x=0, 1) Control Register

Name	Address	Attribute	Initial Value
(2D_A) BGxCNT(x=0, 1)	0x04000008, 0x0400000A	R/W	0x0000
(2D_B) DB_BGxCNT(x=0, 1)	0x04001008, 0x0400100A	R/W	0x0000



- [d15–d14] : Screen Size

Screen Size Settings	Text BG	
	Screen Size	Screen Data Size
00	256x256	2 KB
01	512x256	4 KB
10	256x512	4 KB
11	512x512	8 KB

- [d13] : BG Extended Palette Slot Selection

  This bit specifies the Extended Palette Slot Number used when BG extended palettes are enabled. The settings differ for BG0 and BG1. Extended palettes are enabled/disabled with the DISPCNT [DB_DISPCNT] register. See "3.2.1 VRAM" on page 28 for more information on the palette slot memory map.

  1. BG0CNT [DB_BG0CNT]

0	Slot 0
1	Slot 2

  2. BG1CNT [DB_BG1CNT]

0	Slot 1
1	Slot 3

- SB4-SB0 [d12–d08] : Screen Base Block

  These bits specify (in 2KB units) the starting block in VRAM where screen data is stored. When screen data is actually referenced, the starting address is calculated as follows.

  2D Graphics Engine A

  The starting address is the sum of the DISPCNT register's screen base offset value with a 64KB offset and the screen base block with a 2KB offset.

  (screen base offset x 0x10000) + (screen base block x 0x800)

  2D Graphics Engine B

  The starting address is the screen base block with a 2KB offset.

  (screen base block x 0x800)

- CM [d07] : Color Mode

  This bit specifies whether the screen data references BG character data in 16-color or 256-color format.

0	16-color mode
1	256-color mode

- [d06] : Mosaic

  This bit controls whether the mosaic process for BG is on or off. Set the mosaic size with the MOSAIC [DB_MOSAIC] register.

- CB3-CB0 [d05–d02] : Character Base Block

  These bits specify (in 16KB units) the starting block in VRAM for storing character data. When character data is actually referenced, the starting address is calculated as follows.

  2D Graphics Engine A

  The starting address is the sum of the DISPCNT register's character base offset value with a 64KB offset and the character base block with a 16KB offset.

  (character base offset x 0x10000) + (character base block x 0x4000)

  2D Graphics Engine B

  The starting address is the character base block with a 16KB offset.

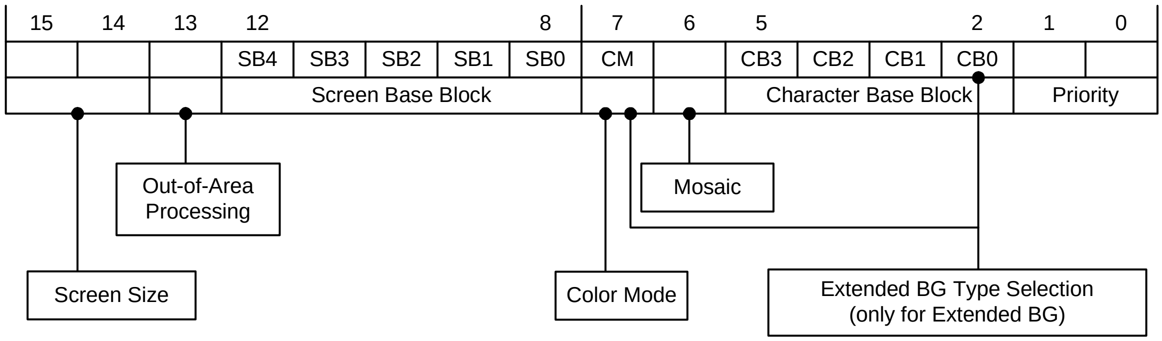
  (character base block x 0x4000)

- [d01–d00] : Priority

  The default order of priority among the BG screens is BG0 > BG1 > BG2 > BG3 (when priority settings are the same). However, this order can be changed. Priorities of 0 (highest) to 3 (lowest) can be set. Be careful of the pixel specifications to which color special effects are applied when changing BG priorities.

BGx(x=2, 3) Control Register

Name	Address	Attribute	Initial Value
(2D_A) BGxCNT(x=2, 3)	0x0400000C, 0x0400000E	R/W	0x0000
(2D_B) DB_BGxCNT(x=2, 3)	0x0400100C, 0x0400100E	R/W	0x0000



The bit definitions for Color Mode, Mosaic, Character Base Block, and Priority are the same as for the BGx (x=0, 1) Control Registers described above. BG2 uses Extended Palette Slot 2 and BG3 uses Extended Palette Slot 3 when extended palettes are enabled. The extended palette numbers used by BG2 and BG3 cannot be changed. The extended palettes can be enabled/disabled with the DISPCNT [DB_DISPCNT] register.

- [d15–d14] : Screen Size

  The screen sizes that can be configured depend on the BG type and are described in Table 6-5 and "Table 6-6 : Screen Sizes (2D Graphics Engine B)" on page 108.

  Note: 2D Graphics Engine A and 2D Graphics Engine B accommodate different combinations of screen sizes and BG types that can be configured. 2D Graphics Engine B cannot set Large-Screen 256-Color Bitmap BG as the BG type. In addition, because a maximum of 128 KB of BG-VRAM can be allocated to 2D Graphics Engine B, screen sizes exceeding 128 KB are prohibited.

Table 6-5 : Screen Sizes (2D Graphics Engine A)

Screen Size Settings	Text BG	Affine BG	Affine Extended BG			Large-Screen 256-Color Bitmap BG
			256-Color x 16-Palette Character BG	256-Color Bitmap BG	Direct-Color Bitmap BG	
00	256x256 (2 KB)	128x128 (256 bytes)	128x128 (512 bytes)	128x128 <16 KB >	128x128 <32 KB >	512x1024 <512 KB >
01	512x256 (4 KB)	256x256 (1 KB)	256x256 (2 KB)	256x256 <64 KB >	256x256 <128 KB >	1024x512 <512 KB >
10	256x512 (4 KB)	512x512 (4 KB)	512x512 (8 KB)	512x256 <128 KB >	512x256 <256 KB >	—
11	512x512 (8 KB)	1024x1024 (16 KB)	1024x1024 (32 KB)	512x512 <256 KB >	512x512 <512 KB >	—

Note: The screen size is enclosed in parentheses () and the bitmap data size in angle brackets <>.

Table 6-6 : Screen Sizes (2D Graphics Engine B)

Screen Size Settings	Text BG	Affine BG	Affine Extended BG		
			256-Color x 16-Palette Character BG	256-Color Bitmap BG	Direct-Color Bitmap BG
00	256x256 (2 KB)	128x128 (256 bytes)	128x128 (512 bytes)	128x128 <16 KB >	128x128 <32 KB >
01	512x256 (4 KB)	256x256 (1 KB)	256x256 (2 KB)	256x256 <64 KB >	256x256 <128 KB >
10	256x512 (4 KB)	512x512 (4 KB)	512x512 (8 KB)	512x256 <128 KB >	Prohibited Setting
11	512x512 (8 KB)	1024x1024 (16 KB)	1024x1024 (32 KB)	Prohibited Setting	Prohibited Setting

Note: The screen size is enclosed in parentheses () and the bitmap data size in angle brackets <>.

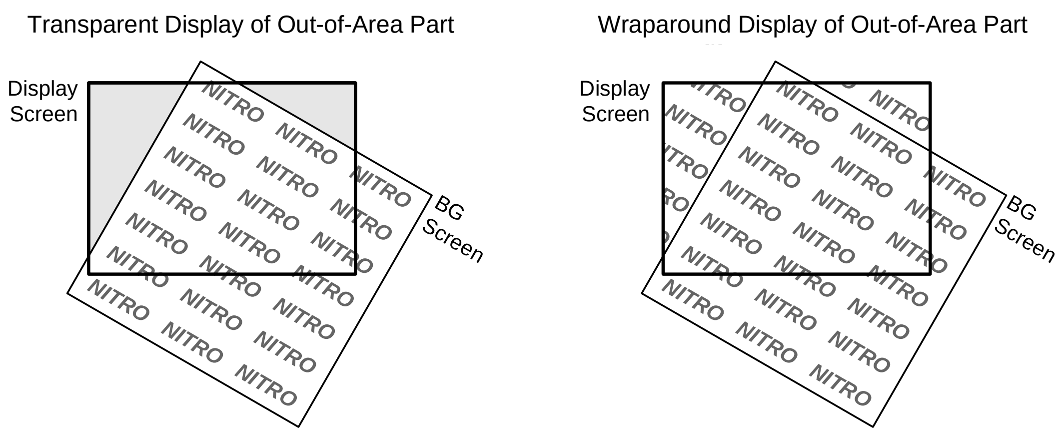
- [d13] : Out-of-Area Processing

  This bit selects either to make out-of-area regions transparent or to wrap around and display when the BG screen does not lie entirely within the display screen because of affine transformations.

0	Transparent display
1	Wraparound display

The difference between the two Out-of-Area processing methods is shown in Figure 6-1.

Figure 6-1 : Out-of-Area Processing Method Differences



- SB4-SB0 [d12–d08] : Screen Base Block

  Calculation of the base address for the screen base block differs according to the BG mode.

1. **Character BG**

   These bits specify (in 2KB units) the starting block in VRAM where screen data is stored. When screen data is actually referenced, the starting address is calculated as follows.

2D Graphics Engine A

The starting address is the sum of the DISPCNT register's screen base offset value with a 64KB offset and the value of the screen base block with a 2KB offset.

(screen base offset x 0x10000) + (screen base block x 0x800)

2D Graphics Engine B

The starting address is the screen base block with a 2KB offset.

(screen base block x 0x800)

2. **256-Color Bitmap BG and Direct Color Bitmap BG**

These bits specify (in 16KB units) the offset address in BG-VRAM where the bitmap data is stored. Because there is no relation to the screen base offset value in the DISPCNT register, the same calculation for the BG bitmap data starting address is used for both 2D Graphics Engine A and 2D Graphics Engine B.

2D Graphics Engine A and 2D Graphics Engine B

(screen base block x 0x4000)

3. **Large-Screen 256-Color Bitmap BG**

The screen base block value is invalid for 2D Graphics Engine A.

The BG mode cannot be set to large-screen 256-color bitmap BG for 2D Graphics Engine B.

- [d07, d02] : Affine Extended BG Type Selection (only with Affine Extended BG)

CM	CB0	Affine Extended BG Type
0	**(See note)**	256-color x 16-palette Character BG
1	**0**	256-color bitmap BG
1	**1**	Direct-color bitmap BG

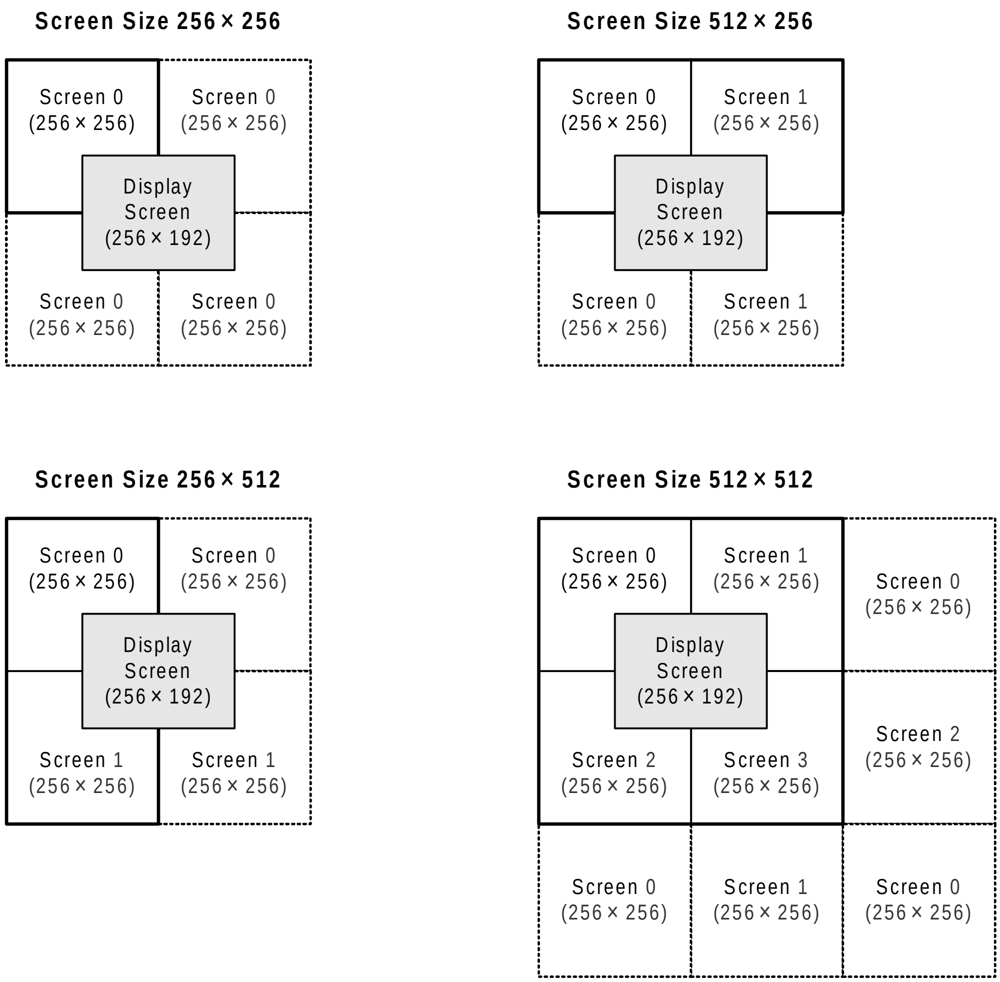
Note: When CM = 0, a unique 256-color x 16-palette Character BG is used and CB3-CB0 are handled as normal character base blocks.

6.2.2.1 Screen Sizes and Display Screens

6.2.2.1.1 Text BG

Text BG screen sizes are shown in Figure 6-2.

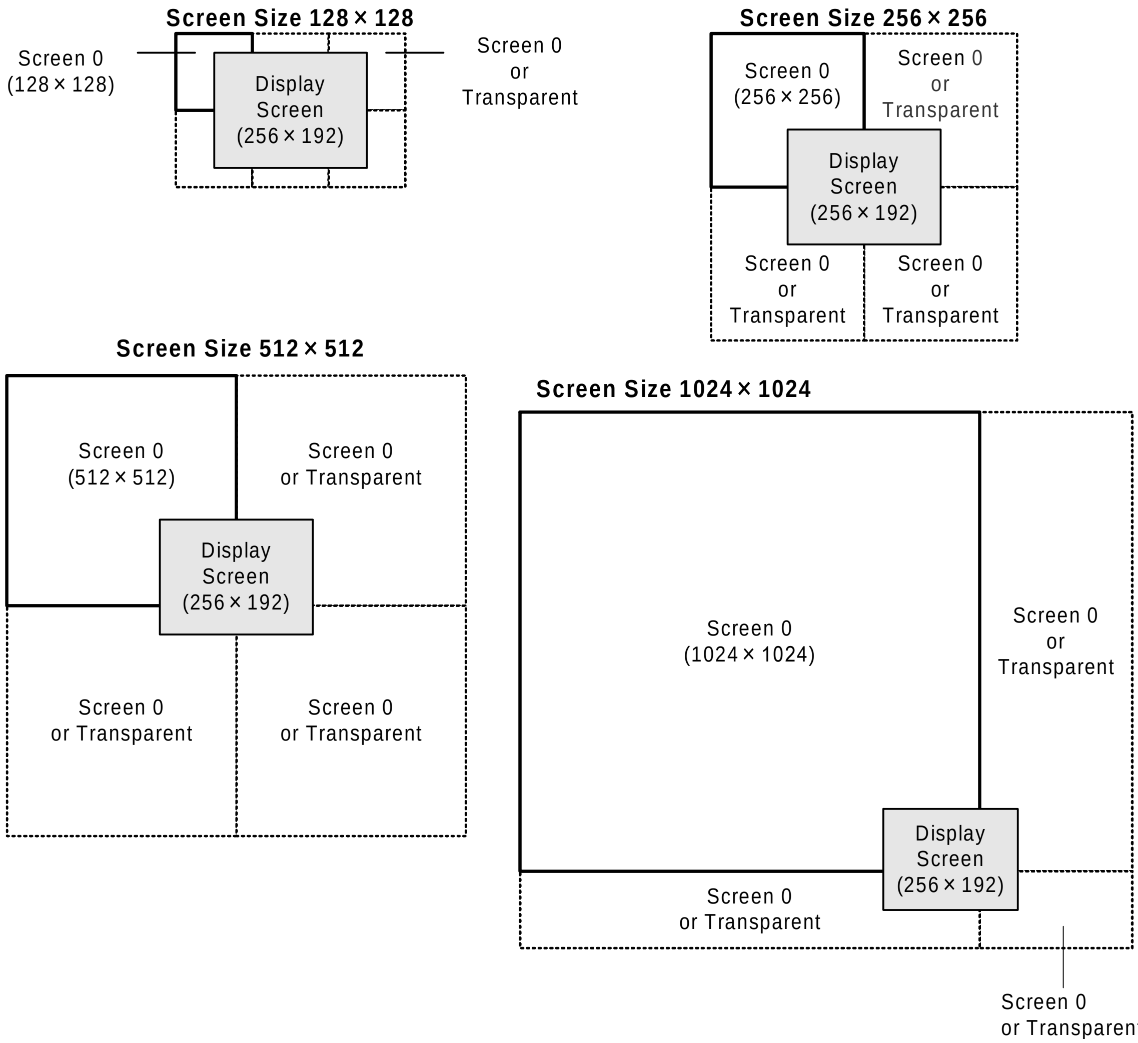
Figure 6-2 : Text BG Screen Size



6.2.2.1.2 Affine BG

Affine BG screen sizes are shown in Figure 6-3.

Figure 6-3 : Affine BG Screen Size



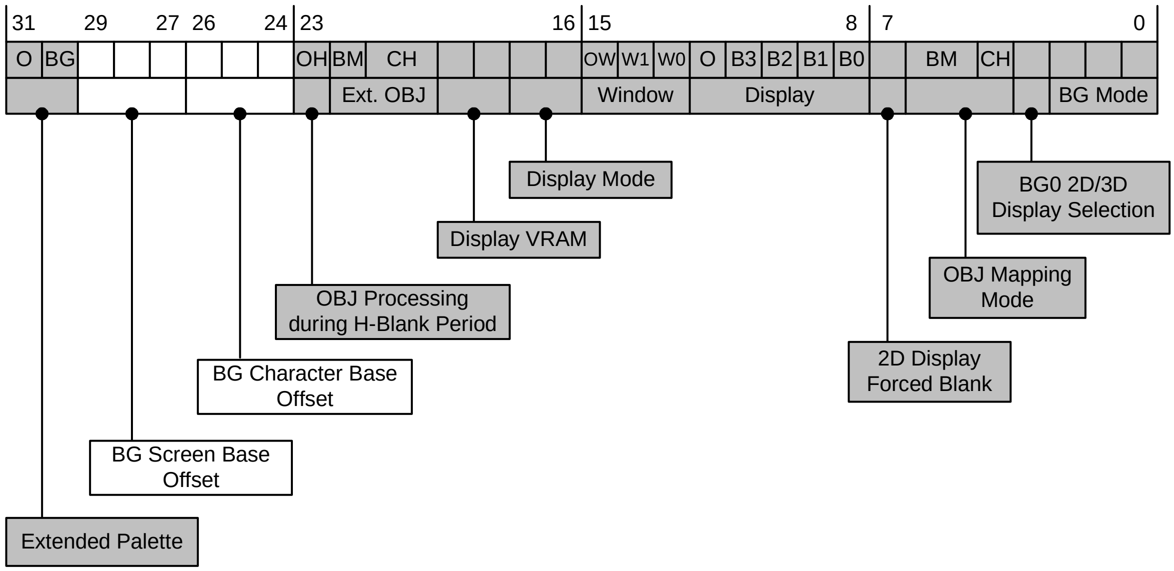
6.2.3 Character BG

For character BG, BG screen composition elements are treated as characters of 8 x 8 dots (8x8-dot). Accordingly, character data is required to display the BG. In addition, character index data for each 8x8-dot unit is required; this character index data is called *screen data*.

Note: 2D Graphics Engine B differs from 2D Graphics Engine A in that there are no settings for the BG screen base offset and BG character base offset.

Display Control Register

Name: DISPCNT Address: 0x04000000 Attribute: R/W Initial Value: 0x00000000



- [d29–d27] : BG Screen Base Offset

  These bits offset (in 64KB units) the base address of the screen data set with the BG Control Register.

  Accordingly, the base address of the BG screen data is calculated as follows:

  The value set in the BG Control Register + (BG screen base offset x 0x10000)

  An arbitrary base address can be specified from a maximum 512 KB of BG-VRAM space.

- [d26–d24] : BG Character Base Offset

  These bits offset (in 64KB units) the base address of the screen data set with the BG Control Register.

  Accordingly, the base address of the BG character data is calculated as follows:

  The value set in the BG Control Register + (BG character base offset x 0x10000)

  An arbitrary base address can be specified from a maximum 512 KB of BG-VRAM space.

6.2.3.1 VRAM Maps of BG Data

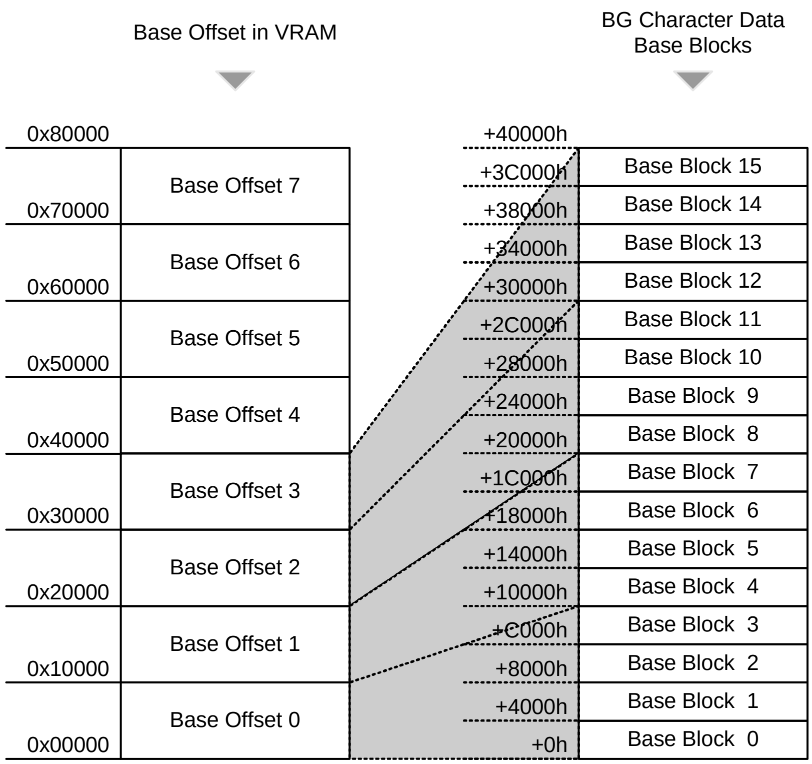
Character BG requires both BG screen data and BG character data. Store both the BG screen data and the BG character data in VRAM that was allocated to BG-VRAM by the RAM Bank Control Register. BG-VRAM can be assigned up to a maximum of 512 KB with 2D Graphics Engine A and up to a maximum of 128 KB with 2D Graphics Engine B.

1. **BG Character Data**

   With 2D Graphics Engine A, the starting address for referencing BG character data can be set by specifying the DISPCNT register's character base offset and the BG Control register's character base block. With 2D Graphics Engine B, there is no setting for character base offset. The VRAM offset for BG character data is shown in Figure 6-4.

   The volume of data depends on the amount of registered character data and the format (Color Mode: 256 or 16 colors).

Figure 6-4 : VRAM Offset for BG Character Data



Note: For 2D Graphics Engine A, the maximum amount of VRAM that can be used for BG character data is 256 KB because the DISPCNT register's base offset cannot be set for each BG screen.

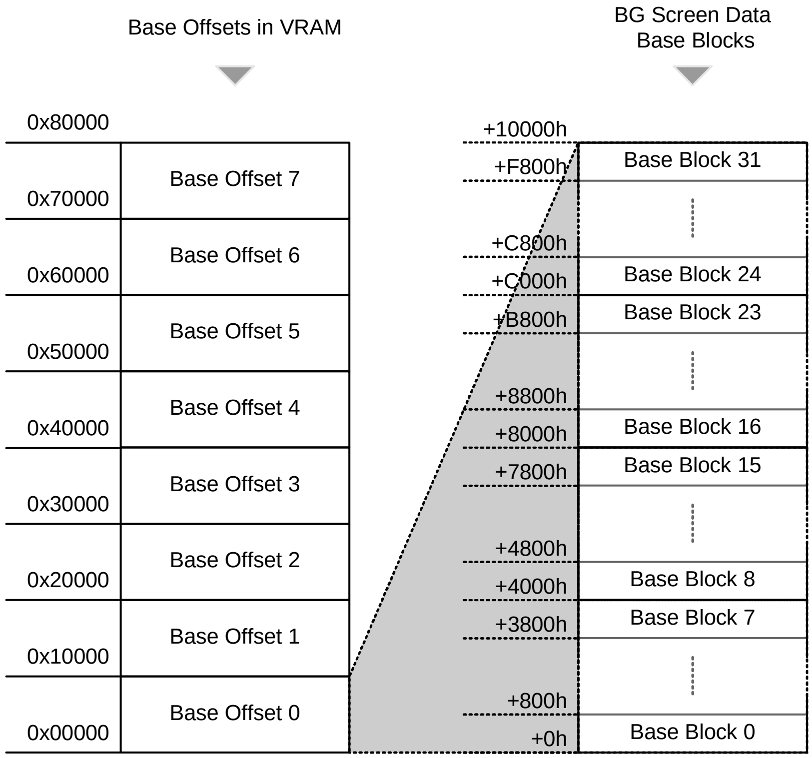
For 2D Graphics Engine B, the maximum amount of VRAM that can be used for BG character data is 128 KB because there are limitations on the size of BG-VRAM that can be allocated.

2. BG Screen Data

For 2D Graphics Engine A, the starting address for referencing BG screen data can be set by specifying the DISPCNT register's screen base offset and the BG Control register's screen base block. With 2D Graphics Engine B, there is no setting for the screen base offset. The VRAM offset for BG screen data is shown in Figure 6-5.

The volume of data depends on the BG type (Text BG or Affine BG) and the screen size.

Figure 6-5 : VRAM Offset for BG Screen Data



Note: For 2D Graphics Engine A, the maximum amount of VRAM that can be used for BG screen data is 64 KB because the DISPCNT register's base offset cannot be set for each BG screen.

With 2D Graphics Engine B, the maximum amount of VRAM that can be used for BG screen data is 64 KB.

6.2.3.2 Text BG

6.2.3.2.1 Screen Data Format

Store the BG screen data starting from the starting address of the BG screen base block specified by the BG Control Register. BG screen data for Text BG screens is configured using the following format:

Text BG Screen Data

15			12	11	10	9	8	7							0
				VF	HF										
Color Palette				Flip		Character Name									

- [d15–d12] : Color Palette

  Palettes applied to characters are specified in the range of 0-15. The Color Palette specification is enabled with 256 colors x 16 palettes or 16 colors x 16 palettes, but it is disabled with 256 colors x 1 palette.

- [d11–d10] : Flip
  - VF: Vertical Flip Flag   HF: Horizontal Flip Flag

0	Do not flip
1	Flip

- [d09–d00] : Character Name

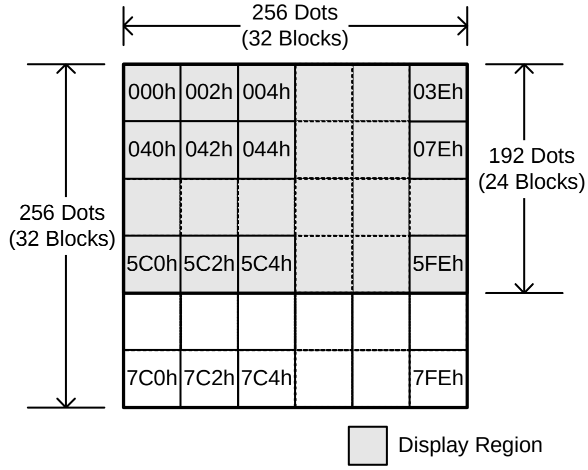
  These bits specify the character number of the character that serves as the origin of the starting address for the character base block specified by the BG Control Register.

6.2.3.2.2 Screen Data Address Mapping

1. 256x256-Dot Screen Size

Figure 6-6 shows the address map for screen data with a 256x256–dot screen size.

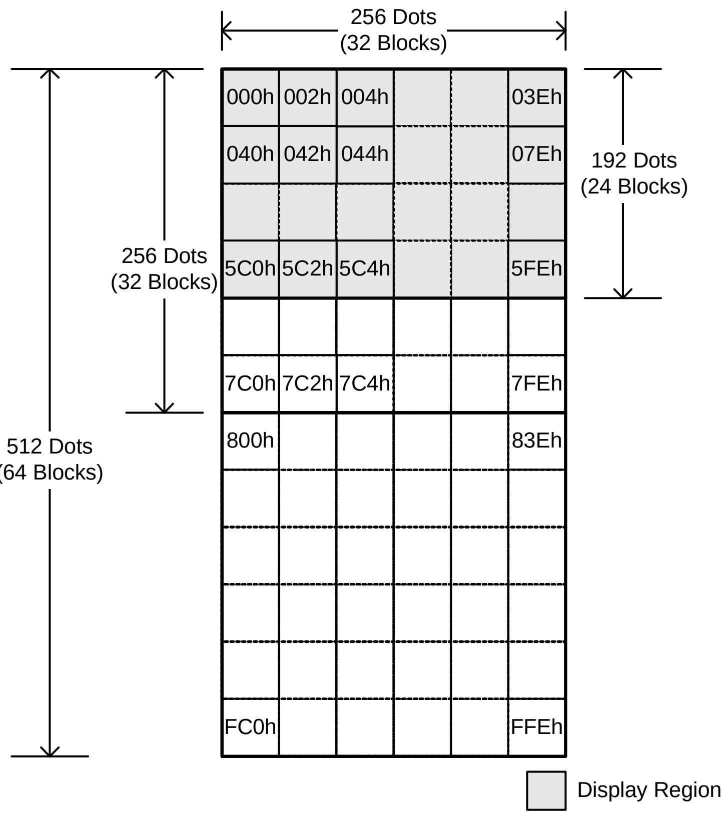
Figure 6-6 : 256x256–Dot Address Mapping (Text BG)



2. 256x512–Dot Screen Size

Figure 6-7 shows the address map for screen data with a 256x512–dot screen size.

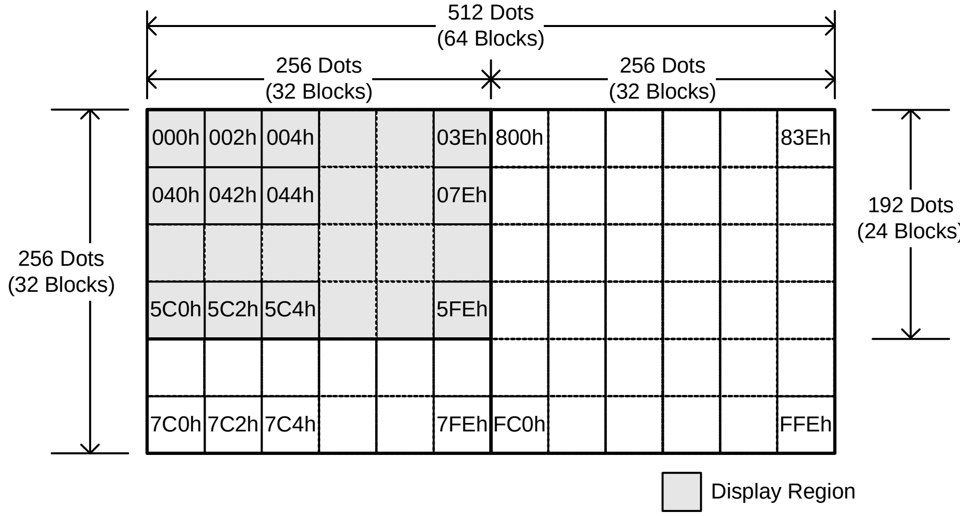
Figure 6-7 : 256x512–Dot Address Mapping (Text BG)



3. 512x256-Dot Screen Size

Figure 6-8 shows the address map for screen data with a 512x256–dot screen size.

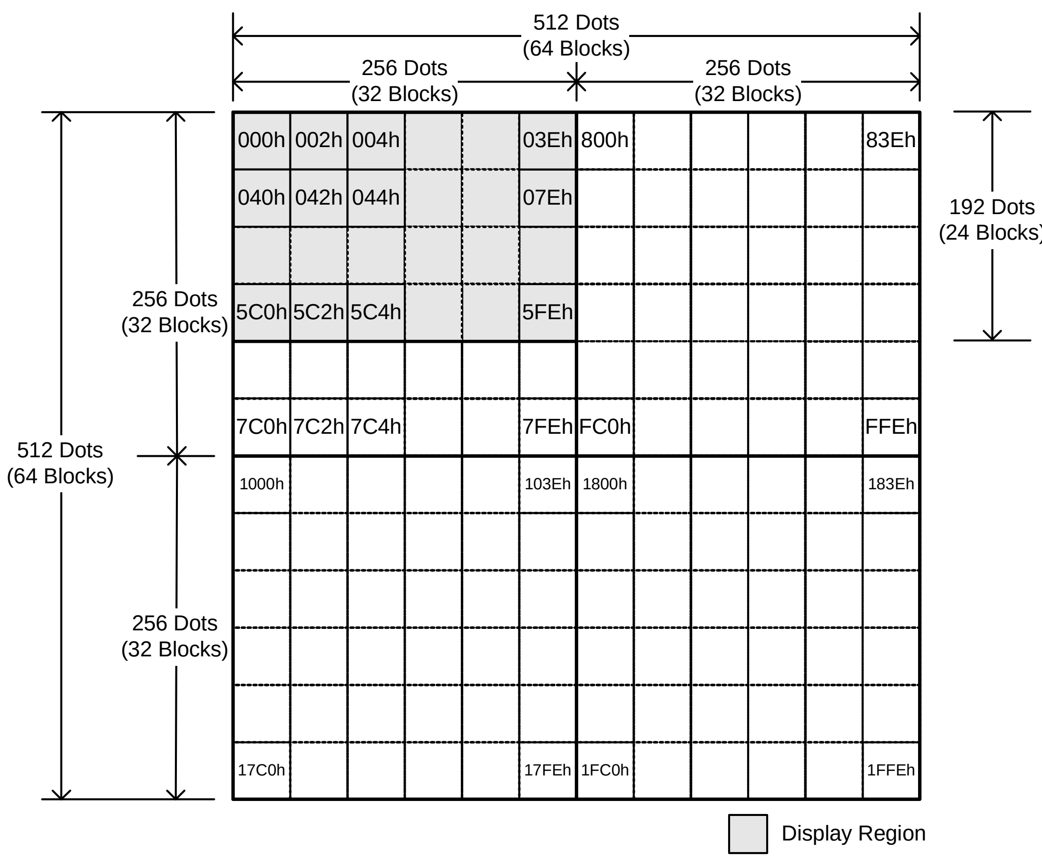
Figure 6-8 : 512x256–Dot Address Mapping (Text BG)



4. 512x512–Dot Screen Size

Figure 6-9 shows the address map for screen data with a 512x512–dot screen size.

Figure 6-9 : 512x512–Dot Address Mapping (Text BG)



6.2.3.2.3 Character Data Formats

The character data formats for Text BG 16-color mode and Text BG 256-color mode are shown below. The Character Display table shows the case when an 8x8–dot character is defined.

6.2.3.2.3.1 16-Color Mode

The character data format for 16-color mode, correspondence between character display and pixel data, and address mapping (Figure 6-10) are shown below.

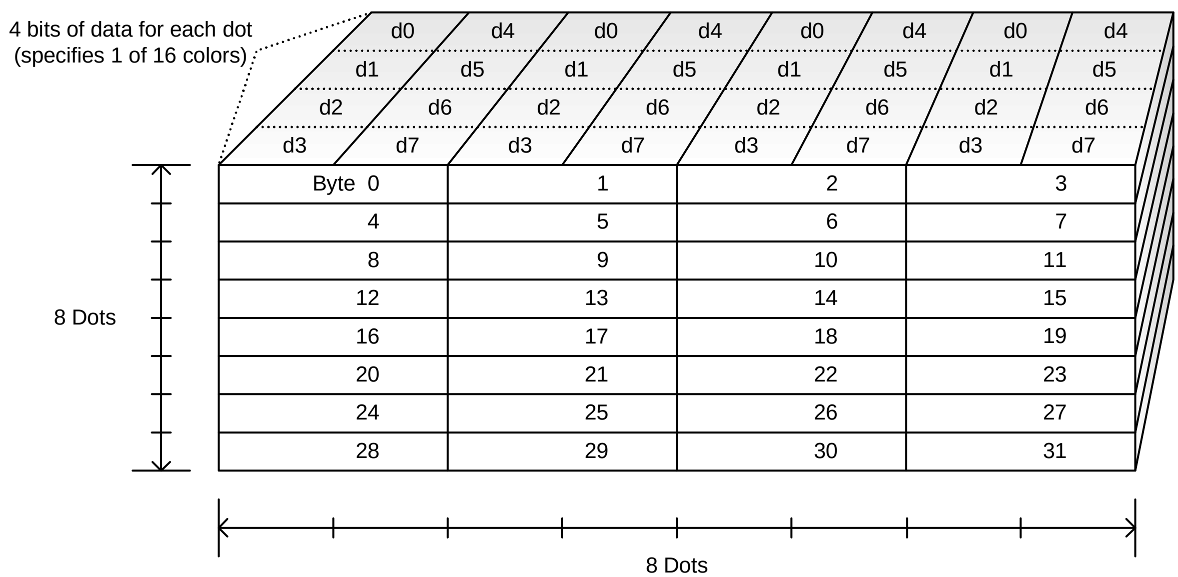
16-Color Mode Character Data Format

15 12	11 8	7 4	3 0
P3	P2	P1	P0
4 pixels worth of data (4 bits/pixel)			

Character Display

P0	P1	P2	P3				

Figure 6-10 : Character Data Address Mapping (Text BG 16-Color Mode)



6.2.3.2.3.2 256-Color Mode

The character data format for 256-color mode, correspondence between character display and pixel data, and address mapping (Figure 6-11) are shown below.

256-Color Mode Character Data Format

15	12	11	8	7	4	3	0
P1				P0			
2 pixels worth of data (8 bits/pixel)							

Character Display

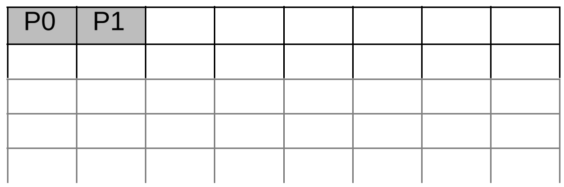
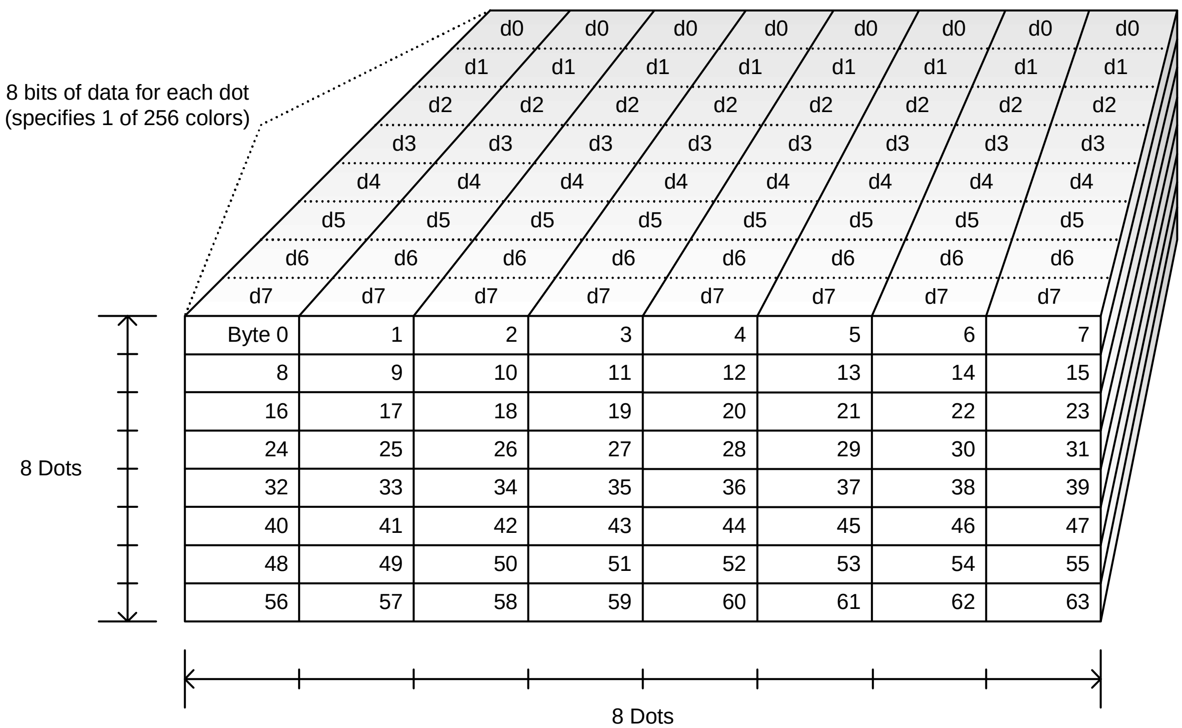


Figure 6-11 : Character Data Address Mapping (Text BG 256-Color Mode)



6.2.3.3 Affine BG

This character format BG type can be rotated and scaled.

Note: Affine BG can be set only with BG2 and BG3. The color mode for Affine BG screens is fixed to 256-color mode. Consequently, the BG Control Register's color-mode setting is disabled. Furthermore, horizontal and vertical flips cannot be performed on Affine BG.

6.2.3.3.1 Screen Data Format

Store the BG screen data starting from the starting address of the BG screen base block specified by the BG Control register.

BG screen data for Affine BG screens is configured using the following format:

Affine BG Screen Data

7							0
Character Name							

- [d07–d00] : Character Name

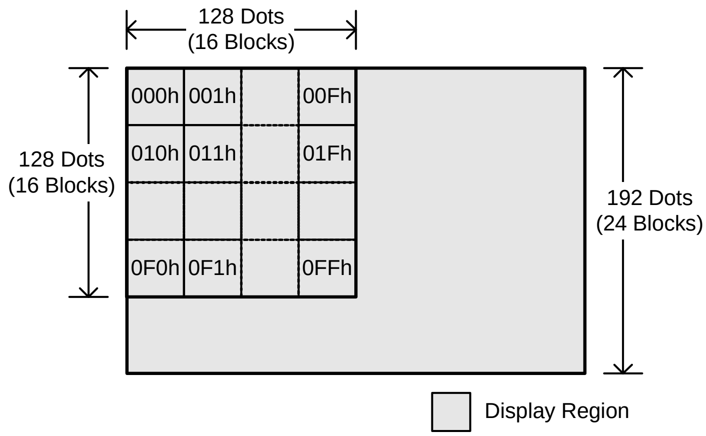
  These bits specify the character number of the character that serves as the origin of the starting address for the character base block specified by the BG Control Register.

6.2.3.3.2 Screen Data Address Mapping

1. 128X128-Dot Screen Size

Figure 6-12 shows the address map for screen data with a 128x128–dot screen size.

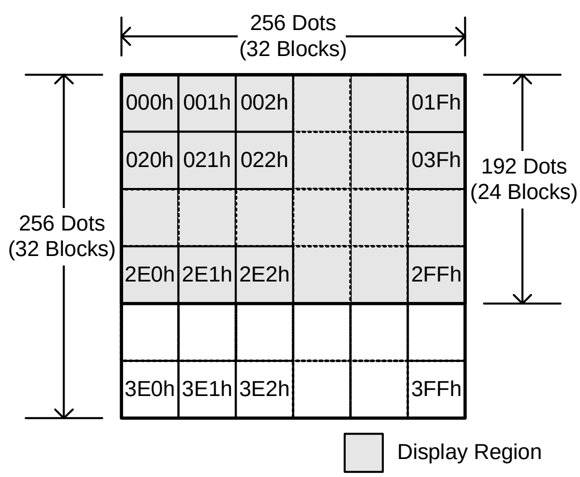
Figure 6-12 : 128X128–Dot Address Mapping (Affine BG)



2. 256x256-Dot Screen Size

Figure 6-13 shows the address map for screen data with a 256x256–dot screen size.

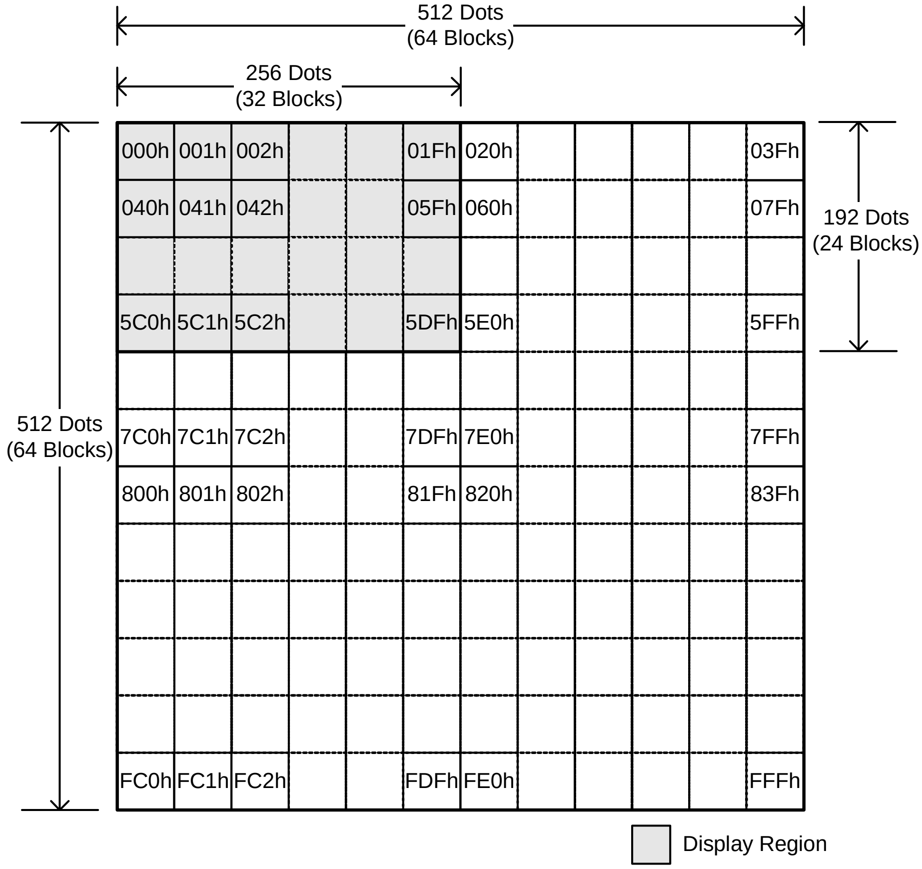
Figure 6-13 : 256x256–Dot Address Mapping (Affine BG)



3. 512x512-Dot Screen Size

Figure 6-14 shows the address map for screen data with a 512x512–dot screen size.

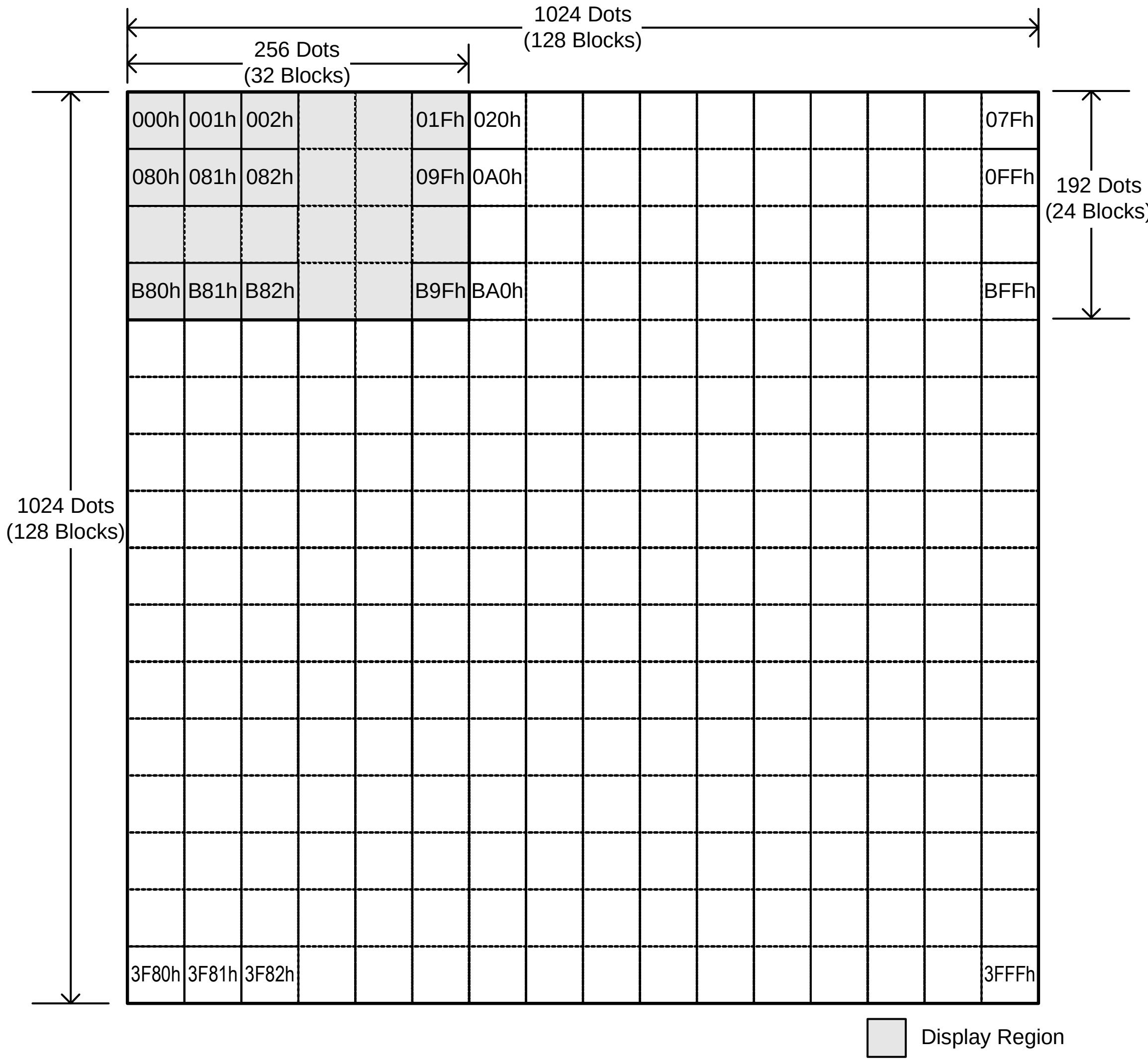
Figure 6-14 : 512x512–Dot Address Mapping (Affine BG)



4. 1024x1024-Dot Screen Size

Figure 6-15 shows the address map for screen data with a 1024x1024–dot screen size.

Figure 6-15 : 1024x1024–Dot Address Mapping (Affine BG)



6.2.3.3.3 Character Data Format

The character data format for Affine BG screens is shown below. The Character Display table shows the case when an 8x8-dot character is defined. Figure 6-16 shows the character data address mapping.

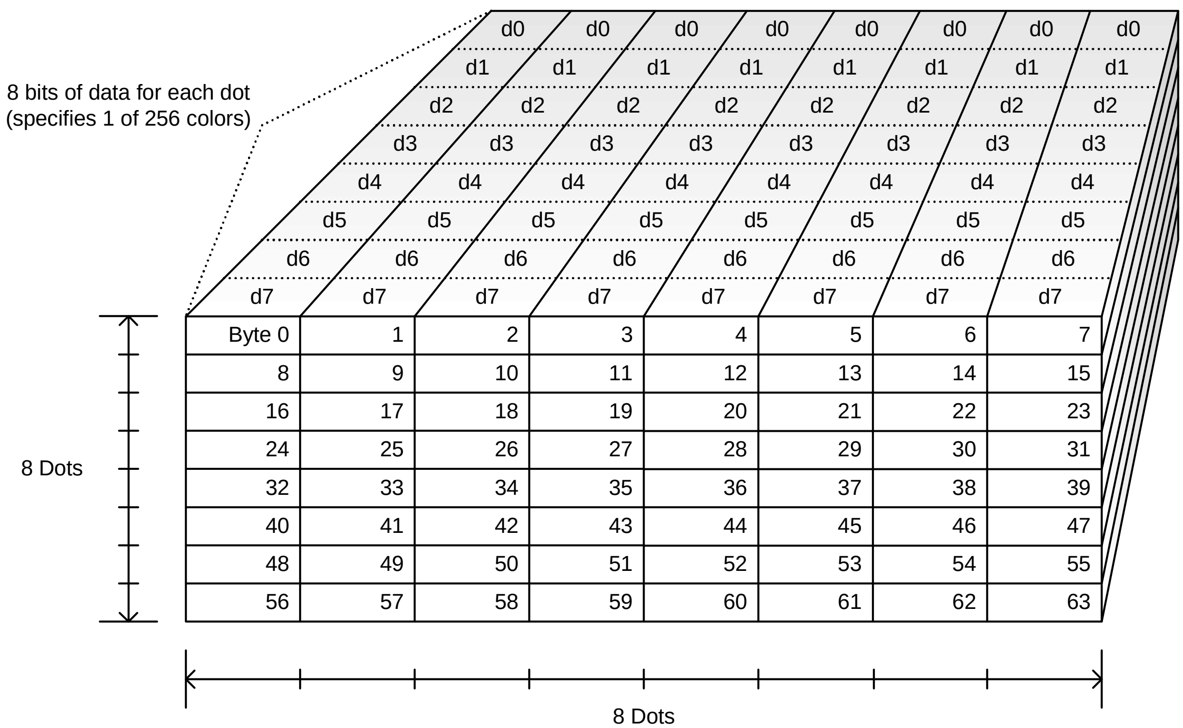
Character Data Format

15	12	11	8	7	4	3	0
P1				P0			
2 pixels worth of data (8 bits/pixel)							

Character Display

P0	P1						

Figure 6-16 : Character Data Address Mapping (Affine BG)



6.2.3.4 256-Color x 16-Palette Character BG (Affine Extended BG)

Select 256-Color x 16-Palette Character BG with the BG Control Register. Select the 256-Color x 16-Palette Character BG by selecting Affine Extended BG as the BG type and setting the BG Control Register Color Mode to 0.

Note: 256-Color x 16-Palette Character BG can be set only for BG2 and BG3.

6.2.3.4.1 Screen Data Format

256-Color x 16-Palette BG Screen Data

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				VF	HF										
Color Palette				Flip		Character Name									

- [d15–d12] : Color Palette

  When enabled, extended palettes applied to characters are specified in the range of 0-15. When extended palettes are disabled, standard palettes are used. Extended palettes are enabled/disabled with the DISPCNT [DB_DISPCNT] Register.

- [d11–d10] : Flip
  - VF: Vertical Flip Flag   HF: Horizontal Flip Flag

0	Do not flip
1	Flip

- [d09–d00] : Character Name

  These bits specify the character number of the character that serves as the origin of the starting address for the character base block specified by the BG Control Register.

6.2.3.4.2 Character Data Format

Character data format is the same as for 256-Color Mode Text BG.

Figure 6-17 shows the character data address mapping.

Character Data Format

15 12	11 8	7 4	3 0
P1		P0	
2 pixels worth of data (8 bits/pixel)			

Character Display

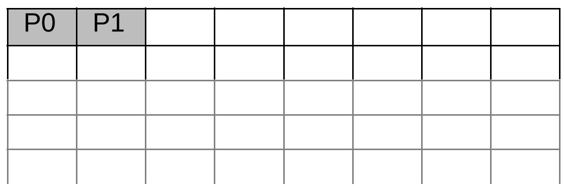
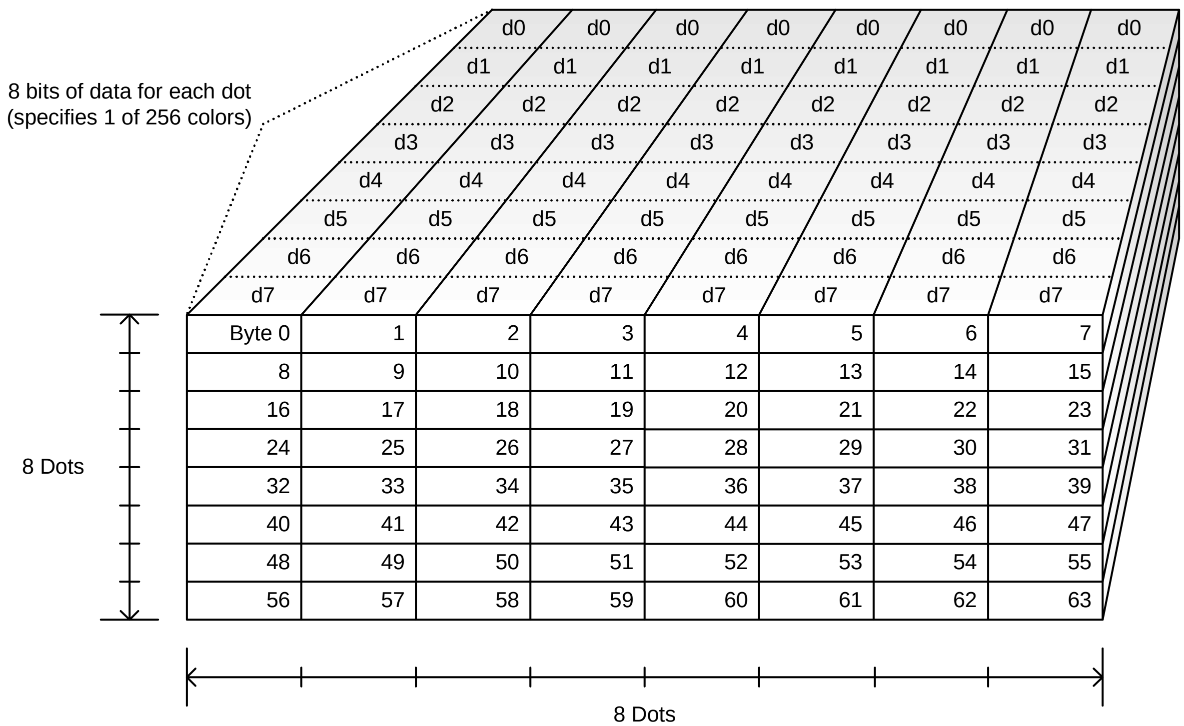


Figure 6-17 : Character Data Address Mapping (256-Color x 16-Palette Character BG)



6.2.4 Bitmap BG

For Bitmap BG, BG screen composition elements are treated as pixels and the contents of VRAM (frame buffer) are displayed as color data for each individual screen pixel.

Note: 256-Color Bitmap BG and Direct-Color Bitmap BG can be set only for BG2 and BG3. Large-Screen 256-Color Bitmap BG can be set only for BG2 with the 2D Graphics Engine A.

6.2.4.1 256-Color Bitmap BG (Affine Extended BG)

Select 256-Color Bitmap BG with the BG Control Register.

Select the 256-Color Bitmap BG by selecting Affine Extended BG as the BG type and setting the BG Control Register Color Mode to 1 and the Character Base Block CB0 to 0.

6.2.4.1.1 Pixel Data Format

The pixel data format for 256-Color Bitmap BG is shown below.

256-Color Bitmap BG Pixel Data Format

7							0
Color Number							

6.2.4.1.2 Pixel Data VRAM Map

The screen base address set in the BG Control Register specifies (in 16KB units) the address in VRAM where the bitmap data is stored. The DISPCNT register's screen base offset value is invalid.

6.2.4.2 Direct-Color Bitmap BG (Affine Extended BG)

Select the Direct-Color Bitmap BG with the BG Control Register.

Direct-Color Bitmap BG can be selected by selecting Affine Extended BG as the BG type and setting the BG Control Register Color Mode to 1 and the Character Base Block CB0 to 1.

6.2.4.2.1 Pixel Data Format

The pixel data format for Direct-Color Bitmap BG is shown below.

Direct-Color Bitmap BG Pixel Data Format

15	14				10	9	8	7		5	4				0
α	BLUE					GREEN					RED				

6.2.4.2.2 Pixel Data VRAM Map

The screen base address set in the BG Control Register specifies (in 16KB units) the address in VRAM where the bitmap data is stored. The DISPCNT register's screen base offset value is invalid.

6.2.4.3 Large-Screen 256-Color Bitmap BG

The Large-Screen 256-Color Bitmap BG cannot be used with 2D Graphics Engine B.

6.2.4.3.1 Pixel Data Format

The pixel data format for Large-Screen 256-Color Bitmap BG is shown below.

Large-Screen 256-Color Bitmap BG Pixel Data Format

7							0
Color Number							

6.2.4.3.2 Pixel Data VRAM Map

The starting address for pixel data is fixed to the starting address of BG-VRAM (0x6000000). Both the BG Control Register's screen base address and the DISPCNT Register's screen base offset value are invalid.

6.2.5 BG Scroll

For each Text BG screen, the display screen and its offset value can be set in dots.

The Offset register is valid only for Text BG. Set the BG reference starting point (see "6.2.6 BG Rotation and Scaling (Affine Transformation)" on page 130) to display Affine BG and Bitmap Mode BG with an offset.

BG Offset Setting Registers

Name	Address	Attribute	Initial Value
(2D_A) BGxOFS(x=0 - 3)	0x04000010, 0x04000014, 0x04000018, 0x0400001C	W	0x00000000
(2D_B) DB_BGxOFS(x=0 - 3)	0x04001010, 0x04001014, 0x04001018, 0x0400101C	W	0x00000000

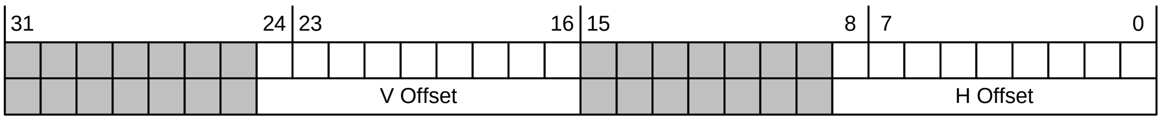
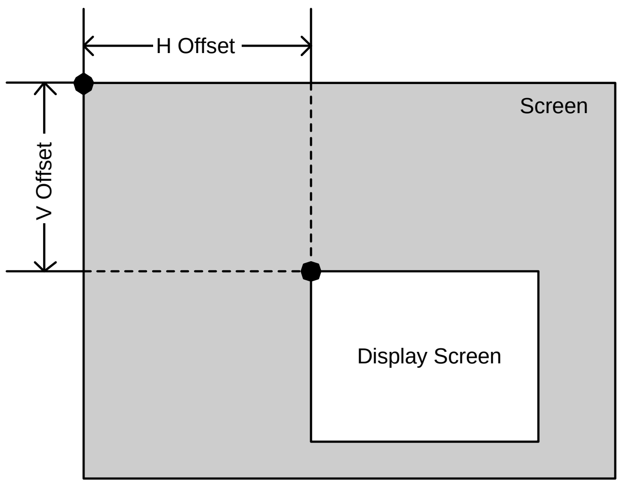


Figure 6-18 shows the offset for BG scrolling.

Figure 6-18 : Offset Schematic

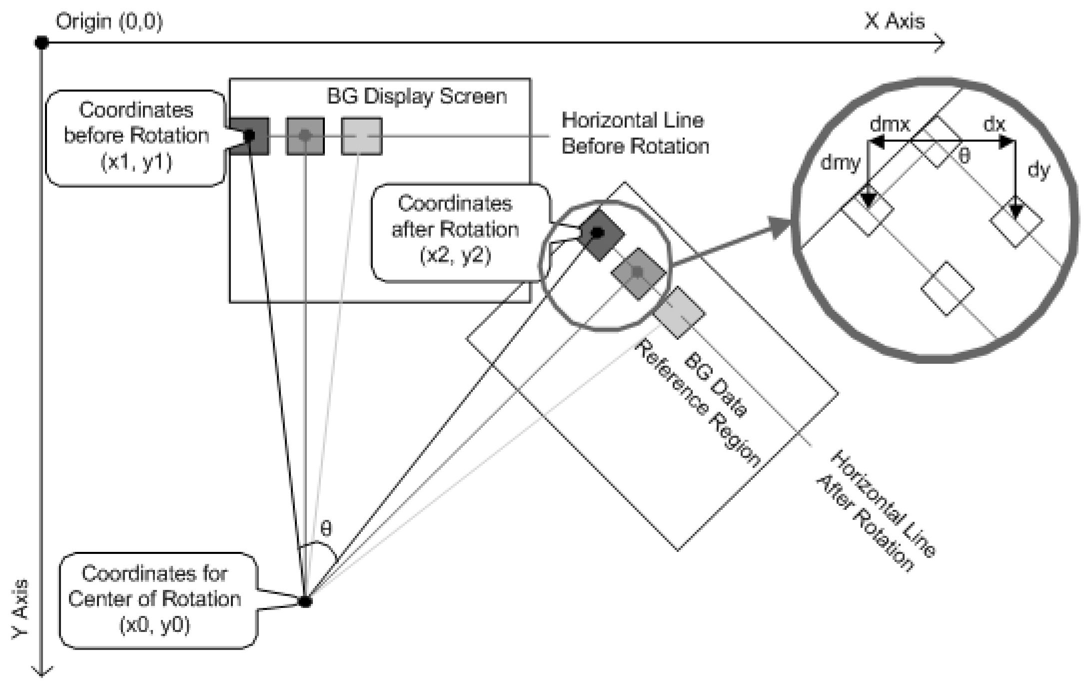


6.2.6 BG Rotation and Scaling (Affine Transformation)

The BG pixels are referenced horizontally in sequence from the top left when a BG is displayed, so a rotated BG can be displayed by rotating the reference direction.

Figure 6-19 shows the rotation and scaling process for a BG.

Figure 6-19 : BG Rotation and Scaling



dx (reference distance in x-direction for same line) = (1/α)cosθ
dy (reference distance in y-direction for same line) = - (1/β)sinθ
dmx (reference distance in x-direction for next line) = (1/α)sinθ
dmy (reference distance in y-direction for next line) = (1/β)cosθ

Note: α is the scale ratio along the x-axis; β is the scale ratio along the y-axis.

The (x2, y2) coordinates correspond to the (x1, y1) coordinates after affine transformation and are calculated with the following formula:

$$\begin{bmatrix} x2 \\ y2 \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} x1 - x0 \\ y1 - y0 \end{bmatrix} + \begin{bmatrix} x0 \\ y0 \end{bmatrix}$$

$$A = \frac{1}{\alpha}\cos\theta,\ B = \frac{1}{\alpha}\sin\theta,\ C = -\frac{1}{\beta}\sin\theta,\ D = \frac{1}{\beta}\cos\theta$$

- BG Rotation and Scaling Process

  1. Using the equation above, calculate the results of affine transformation for the top-left coordinates of the display screen, and then set the results as the BG data reference start point in the following registers:
     - 2D Graphics Engine A BGxX, BGxY Registers (x=2, 3)
     - 2D Graphics Engine B DB_BGxX, DB_BGxY Registers (x=2, 3)

     Also, refer to "Figure 6-19 : BG Rotation and Scaling" on page 130, and set the BG data reference direction in the following registers:
     - 2D Graphics Engine A BGxPA, BGxPB, BGxPC, BGxPD Registers (x=2, 3)
     - 2D Graphics Engine B DB_BGxPA, DB_BGxPB, DB_BGxPC, DB_BGxPD Registers (x=2, 3)
  2. The image processing circuitry sums the cumulative increase in the x-direction (dx and dy) and calculates the x-direction coordinates in relation to the BG data reference start point set in these registers.
  3. If the line advances, the rendering start point coordinates for the next line are calculated by summing the cumulative increase in the y-direction (dmx and dmy) in relation to the reference start point. Then the process in Step 2 is performed.
  4. If the BG Data Reference Start Point Registers are overwritten during an H-Blank, the cumulative sum for the y-direction related to those registers is not computed. Use this mode to have the CPU change the affine transformation parameters and the center coordinates for each line.

BG Data Reference Start Point Setting Registers

Name	Address	Attribute	Initial Value
(2D_A) BGxX (x=2, 3)	0x04000028, 0x04000038	W	0x00000000
(2D_B) DB_BGxX (x=2, 3)	0x04001028, 0x04001038	W	0x00000000

31–28	27	26–8	7–0
	S	INTEGER_SX	DECIMAL_SX
	x-Coordinate of the Reference Start Point (Affine Transformation Result)		

Name	Address	Attribute	Initial Value
(2D_A) BGxY (x=2, 3)	0x0400002C, 0x0400003C	W	0x00000000
(2D_B) DB_BGxY (x=2, 3)	0x0400102C, 0x0400103C	W	0x00000000

31–28	27	26–8	7–0
	S	INTEGER_SY	DECIMAL_SY
	y-Coordinate of the Reference Start Point (Affine Transformation Result)		

BG Data Reference Direction Setting Registers

Name	Address	Attribute	Initial Value
(2D_A) BGxPA (x=2, 3)	0x04000020, 0x04000030	W	0x0100
(2D_B) DB_BGxPA (x=2, 3)	0x04001020, 0x04001030	W	0x0100

15	8	7 0
S	INTEGER_DX	DECIMAL_DX
Reference distance dx in x-direction for the same line		

Name	Address	Attribute	Initial Value
(2D_A) BGxPB (x=2, 3)	0x04000022, 0x0400032	W	0x0000
(2D_B) DB_BGxPB (x=2, 3)	0x04001022, 0x0401032	W	0x0000

15	8	7 0
S	INTEGER_DMX	DECIMAL_DMX
Reference distance dmx in x-direction for the next line		

Name	Address	Attribute	Initial Value
(2D_A) BGxPC (x=2, 3)	0x04000024, 0x0400034	W	0x0000
(2D_B) DB_BGxPC (x=2, 3)	0x04001024, 0x0401034	W	0x0000

15	8	7 0
S	INTEGER_DY	DECIMAL_DY
Reference distance dy in y-direction for the same line		

Name	Address	Attribute	Initial Value
(2D_A) BGxPD(x=2, 3)	0x04000026, 0x0400036	W	0x0100
(2D_B) DB_BGxPD(x=2, 3)	0x04001026, 0x0401036	W	0x0100

15	8	7 0
S	INTEGER_DMY	DECIMAL_DMY
Reference distance dmy in y-direction for the next line		

6.3 OBJ

TWL can handle two types of OBJ: Character OBJ and Bitmap OBJ. Table 6-7 summarizes the features for both Character OBJ and Bitmap OBJ.

Table 6-7 : OBJ Overview

Item	Character OBJ	Bitmap OBJ
Number of Display Colors	1. Standard Palette 16 colors x 16 palettes 256 colors x 1 palette 2. Extended Palette 16 colors x 16 palettes (Standard Palette) 256 colors x 16 palettes (Extended Palette)	32,768 colors
Number of Characters (Converted to 8x8-Dot)	1. One-dimensional Mapping 1,024 to 8,192 (16-color mode) 512 to 4,096 (256-color mode) 2. Two-dimensional Mapping 1,024 (16-color mode) 512 (256-color mode)	1. 1D Mapping 1,024 to 2,048 2. 2D Mapping 256
Character Size	8x8 dot to 64x64 dot (12 varieties)	
Maximum Number Displayed on One Screen	128 (converted to 64x64 dot)	
Maximum Number Displayed on One Line	128 (converted to 8x8 dot)	
Features	HV Offset, HV Flip, Affine Transformation, Translucence (see note), Mosaic, and Priority settings	

Note: See "6.7 Color Special Effects" on page 170 to learn about OBJ color effects.

- Other items

  See "6.6 Windows" on page 166 to learn about OBJ windows.

- Number of OBJ that can be displayed on one line

  Table 6-7 gives the capacity of OBJ that can be displayed on one line under the most efficient conditions. When display OBJ are positioned in series from the start of OAM, the number of OBJ that can be displayed on one line is calculated as follows:

  (H dot count x 6 – 6)/rendering cycle count) = Number of OBJ displayable on 1 line (128 max.)

  H dot count is normally 355 dots, but it becomes 256 dots if the DISPCNT [DB_DISPCNT] Register's OBJ Processing during H-Blank Period flag is set to 1 (see "5.2 LCD" on page 73).

  x 6 represents the number of cycles that the OBJ rendering circuitry can use per dot. *- 6* represents the number of cycles needed for the OBJ rendering pre-process at the start of the H-line.

  Table 6-8 shows the relation between the *rendering cycle count* and the number of OBJ that can be displayed on one line.

Table 6-8 : Rendering Cycle Count and Number of OBJ Displayable on One Line

OBJ H-Size	Render Cycle Count		Number of OBJ Displayable on One Line	
	Normal OBJ*1	Affine OBJ*2	Normal OBJ	Affine OBJ
8	8	26	128	81
16	16	42	128	50
32	32	74	66	28
64	64	138	33	15
128 (Double-Sized 64)	—	266	—	7

Note 1: A Normal OBJ has the OBJ Mode of OBJ Attribute 0 set to Normal OBJ.

Note 2: An Affine OBJ has the Affine Enable Flag of OBJ Attribute 0 set to Enable.

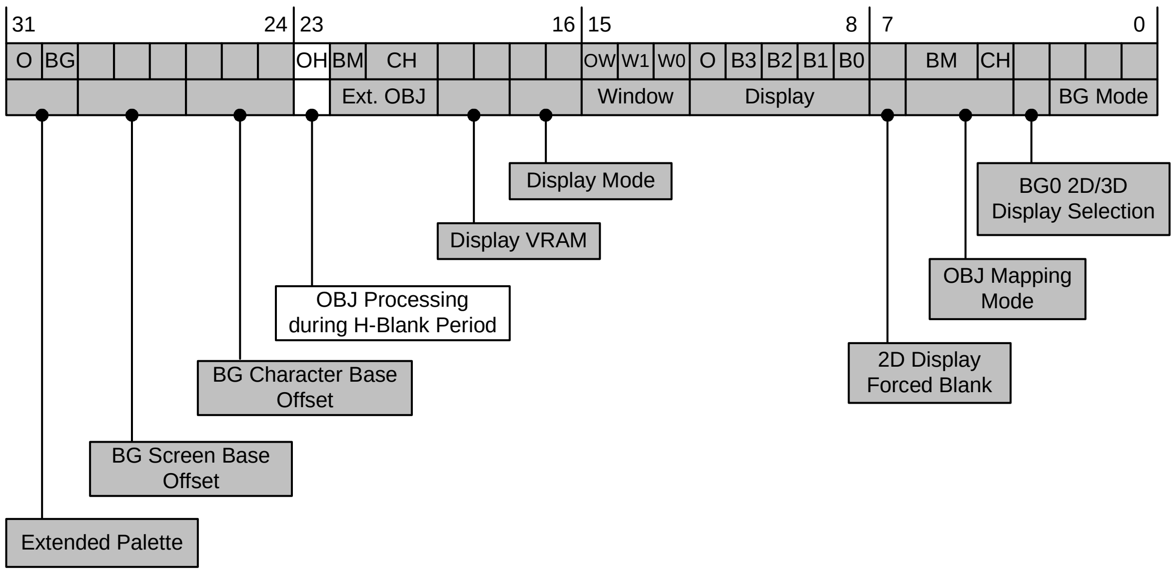
Table 6-8 shows values under the most efficient conditions. Efficiency is actually lower because some OBJ in OAM are outside of the rendered line. Two cycles are lost for an OBJ outside of the rendered line.

6.3.1 OBJ Display Control

The overall configuration of OBJ features is performed with the DISPCNT Register for 2D Graphics Engine A and with the DB_DISPCNT Register for 2D Graphics Engine B. The settings for an individual OBJ are configured with the OBJ Attribute Data stored in OAM. (This subject is touched on later.)

Display Control Register (2D Graphics Engine A)

Name: DISPCNT Address: 0x04000000 Attribute: R/W Initial Value: 0x00000000



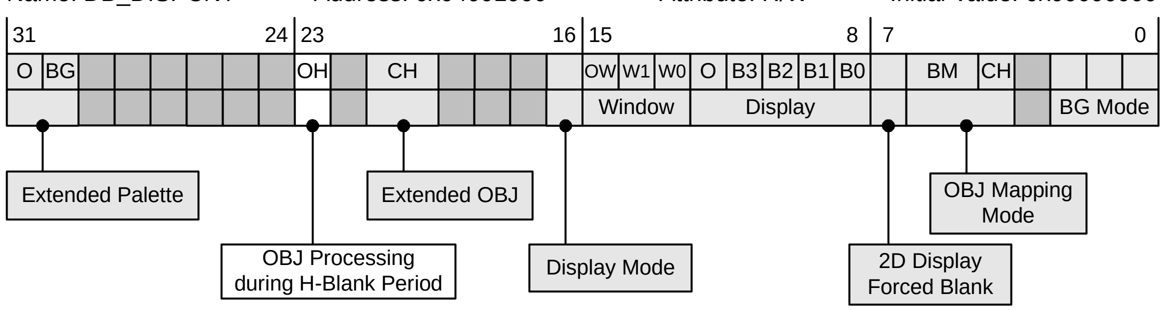
- OH [d23] : OBJ Processing during H-Blank Period Flag

  When set to 0, the OBJ render process is performed during the entire H-line period (including the H-Blank period).

  When set to 1, the OBJ render process is performed only during the display period, but not during the H-Blank period. In this case, the maximum number of OBJ cannot be displayed.

Display Control Register 1 (2D Graphics Engine B)

Name: DB_DISPCNT Address: 0x04001000 Attribute: R/W Initial Value: 0x00000000



- OH [d23] : OBJ Processing during H-Blank Period Flag

  When set to 0, the OBJ render process is performed during the entire H-line period (including the H-Blank period).

  When set to 1, the OBJ render process is performed only during the display period, but not during the H-Blank period. In this case, the maximum number of OBJ cannot be displayed.

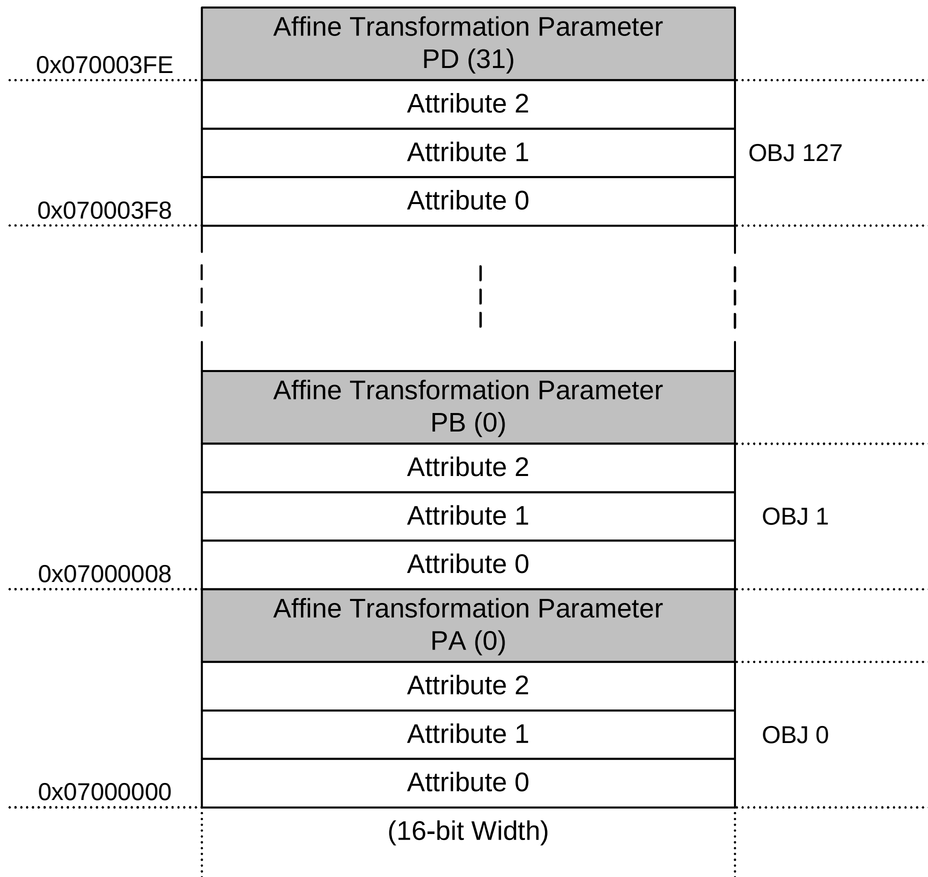
6.3.2 OAM

An OBJ is displayed by storing data in Object Attribute Memory (OAM). A total of 128 sets of OBJ data can be written to the TWL Processor's internal OAM (1 KB from 0x07000000 – 0x070003FF for 2D Graphics Engine A and 1 KB from 0x7000400 – 0x70007FF for 2D Graphics Engine B). Accordingly, a total of 128 OBJ characters of any size can be displayed on the LCD.

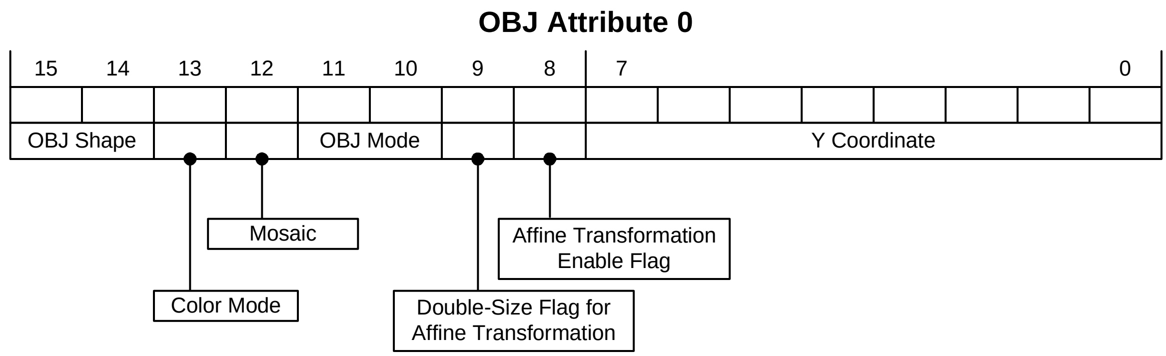
6.3.2.1 Memory Map

48 bits x 128 sets of OBJ Attribute data can be written to OAM. If an OBJ is to be rotated and scaled, a total of 32 groups of affine transformation parameters PA, PB, PC, and PD can be written to OAM as shown in Figure 6-20.

Figure 6-20 : OAM Memory Map (Add 0x400h to 2D Graphics Engine B Addresses)



6.3.2.2 OAM Data Format



- [d15–d14] : OBJ Shape

  These bits set the shape of the OBJ. The number of dots in the OBJ's horizontal and vertical direction is determined by this setting and the OBJ size specification in OBJ Attribute 1. See OBJ Size under "OBJ Attribute 1" on page 139.

00	Square
01	Long rectangle
10	Tall rectangle
11	Prohibited code

- [d13] : Color Mode

  This bit sets whether the OBJ character data is referenced in 16-color format or 256-color format. Be sure to set Color Mode to 0 for Direct-Color Bitmap OBJ settings.

0	16-color mode
1	256-color mode

- [d12] : Mosaic

0	Mosaic off
1	Mosaic on

- [d11–d10] : OBJ Mode

  The 00 to 10 settings specify Character OBJ. When 10 (the OBJ Window) is specified, data is not displayed as normal OBJ; if there are dots of non-zero character data, the data is handled as an OBJ window that can take any shape. See "6.6 Windows" on page 166 for display settings inside the OBJ window. The 11 setting specifies Bitmap OBJ.

00	Normal OBJ
01	Translucent OBJ
10	OBJ Window
11	Bitmap OBJ

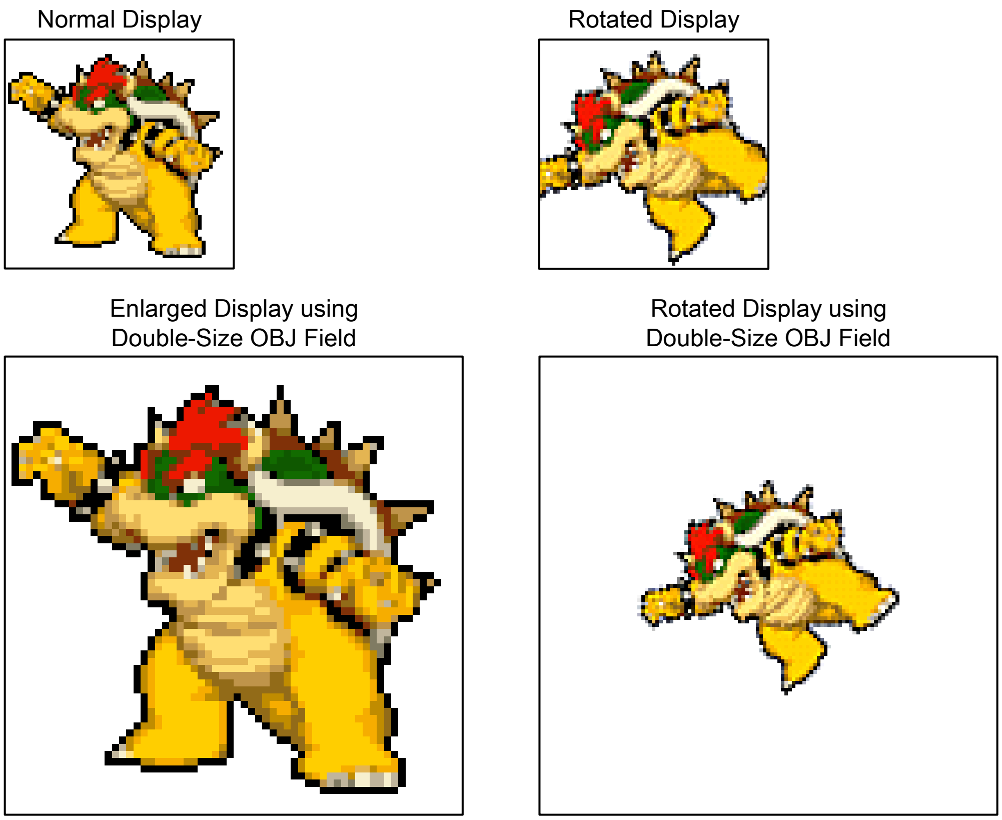
- [d09] : Double-Size Flag for Affine Transformation

0	Disable double-size
1	Enable double-size

When the d08 Affine Transformation Enable Flag is set to 1, the OBJ field can be doubled in size for display. Using a double-size OBJ field allows a rotated OBJ to be displayed in its entirety, without losing any sections. In addition, an OBJ can be enlarged up to double its original size and displayed without losing any sections (see Figure 6-21).

When the d08 Affine Transformation Enable Flag is set to 0 and this bit to 1, the OBJ is hidden.

Figure 6-21 : Affine Transformation of Double-Size OBJ Field



- [d08] : Affine Transformation Enable Flag

When this bit is enabled, the affine transformation parameters set in OBJ Attribute 1 are referenced.

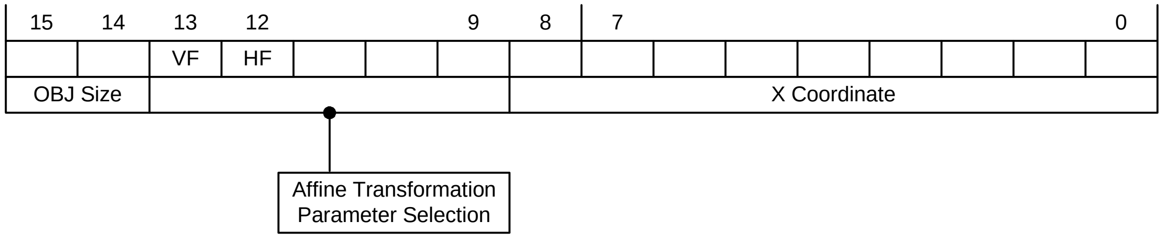
When this bit is disabled and the d09 Double-Size Flag for Affine Transformation is set to 1, the OBJ is hidden.

0	Disable
1	Enable

- [d07–d00] : Y Coordinate

These bits specify the OBJ's Y coordinate in the display screen in the range of 0 to 255.

OBJ Attribute 1



- [d15–d14] : OBJ Size

  These bits set the OBJ size. The number of vertical and horizontal dots for an OBJ depends on this setting and the OBJ shape set in OBJ Attribute 0. Table 6-9 shows the relationship.

Table 6-9 : OBJ Shape and OBJ Size Settings

OBJ Attribute 1 OBJ size OBJ Attribute 0 OBJ shape	00	01	10	11
00 (Square)	8x8	16x16	32x32	64x64
01 (Long rectangle)	16x8	32x8	32x16	64x32
10 (Tall rectangle)	8x16	8x32	16x32	32x64
11 (Prohibited setting)	Prohibited Setting			

- [d13–d09] : Affine Transformation Parameter Selection

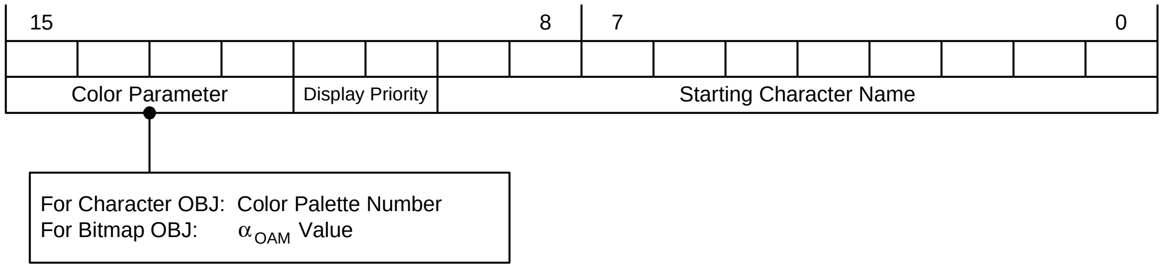
  These bits specify which of the 32 sets of OAM PA – PD affine transformation parameters to reference.

  When the OBJ Attribute 0 Affine Transformation Enable Flag is set to 0 (disabled), the [d13] VF bit is treated as the Vertical Flip Flag and the [d12] HF bit is treated as the Horizontal Flip Flag.

- [d08–d00] : x Coordinate

  These bits specify the OBJ's x-coordinate in the display screen in the range of 0 to 511.

OBJ Attribute 2



- [d15–d12] : Color Parameter

  What these bits specify depends on whether OBJ Mode in OBJ Attribute 0 is set to Character OBJ or Bitmap OBJ. These bits specify the Color Palette Number for Character OBJ. They specify the α_{OAM} for Bitmap OBJ. The α_{OAM} value is used as a factor for blending with the BG for the Bitmap OBJ (see "6.3.4 Bitmap OBJ" on page 151).

  1. For Character OBJ: Color Palette Number

     This specifies one of 16 palettes to apply to the character data.

     This bit is invalid in 256-color mode when extended palettes are disabled (see "OBJ Attribute 0" on page 137).

     Extended palettes are enabled/disabled with the DISPCNT [DB_DISPCNT] register.

  2. For Bitmap OBJ: α_{OAM} value

     The α_{OAM} value is an element of the OBJ's transparency α, where $\alpha = \alpha_{BMP}$ x $(\alpha_{OAM} +1)$. Set α_{BMP} using the Bitmap OBJ data (see "6.3.4.1 Bitmap OBJ Data" on page 153).

- [d11–d10] : Display Priority

  These bits set the order of priority for display.

  See "6.9 Display Priority" on page 175 to learn about the priority relation with BG.

  The priority set with this bit is invalid when OBJ Mode in OAM Attribute 0 is set to OBJ Window. See "6.6 Windows" on page 166 to learn about the precedence of windows.

- [d09–d00] : Starting Character Name

  The basic character number at the start of the OBJ character data mapped in OBJ-VRAM is written here. The specification in Bitmap OBJ mode is the same as for Character mode with 8x8-dot units.

  2D Mapping Mode

  When in 2D Mapping mode and 256-Color mode, the starting character name's lowest bit is fixed at 0. In addition, OBJ-VRAM references regions only up to 32 KB.

  1D Mapping Mode

  When in 1D Mapping mode, the capacity of OBJ-VRAM can be expanded (see the RAM Bank Control Registers 0 and 1 and the Display Control Register).

The boundary of the starting character name varies, as shown in Table 6-10 and Table 6-11, depending on the OBJ-VRAM capacity to allow the entire OBJ-VRAM region to be referenced with the setting region of the starting character name (10 bits).

Table 6-10 : Character OBJ

OBJ-VRAM Capacity	Starting Character Name Boundary
32 KB	32 bytes
64 KB	64 bytes
128 KB	128 bytes
256 KB	256 bytes

Table 6-11 : Bitmap OBJ

OBJ-VRAM Capacity	Starting Character Name Boundary
128 KB	128 bytes
256 KB	256 bytes

Note: The maximum capacity of OBJ-VRAM is 128 KB for 2D Graphics Engine B because of restrictions on VRAM allocation. Therefore, the Character OBJ and Bitmap OBJ capacity cannot be set to 256 KB.

Affine Transformation Parameters

See "6.3.2.3 OBJ Rotation and Scaling (Affine Transformation)" on page 143 for how to determine the OBJ's affine transformation parameters.

Affine Transformation Parameter PA

15	14 ... 8	7 ... 0
S_PA	INTEGER_PA	DECIMAL_PA
Distance dx moved in x-direction on the same line		

Signed fixed-point decimal (sign + 7-bit integer + 8-bit decimal part)

Affine Transformation Parameter PB

15	14 ... 8	7 ... 0
S_PB	INTEGER_PB	DECIMAL_PB
Distance dmx moved in x-direction on the next line		

Signed fixed-point decimal (sign + 7-bit integer + 8-bit decimal part)

Affine Transformation Parameter PC

15	14 ... 8	7 ... 0
S_PC	INTEGER_PC	DECIMAL_PC
Distance dy moved in y-direction on the same line		

Signed fixed-point decimal (sign + 7-bit integer + 8-bit decimal part)

Affine Transformation Parameter PD

15	14 ... 8	7 ... 0
S_PD	INTEGER_PD	DECIMAL_PD
Distance dmy moved in y-direction on the next line		

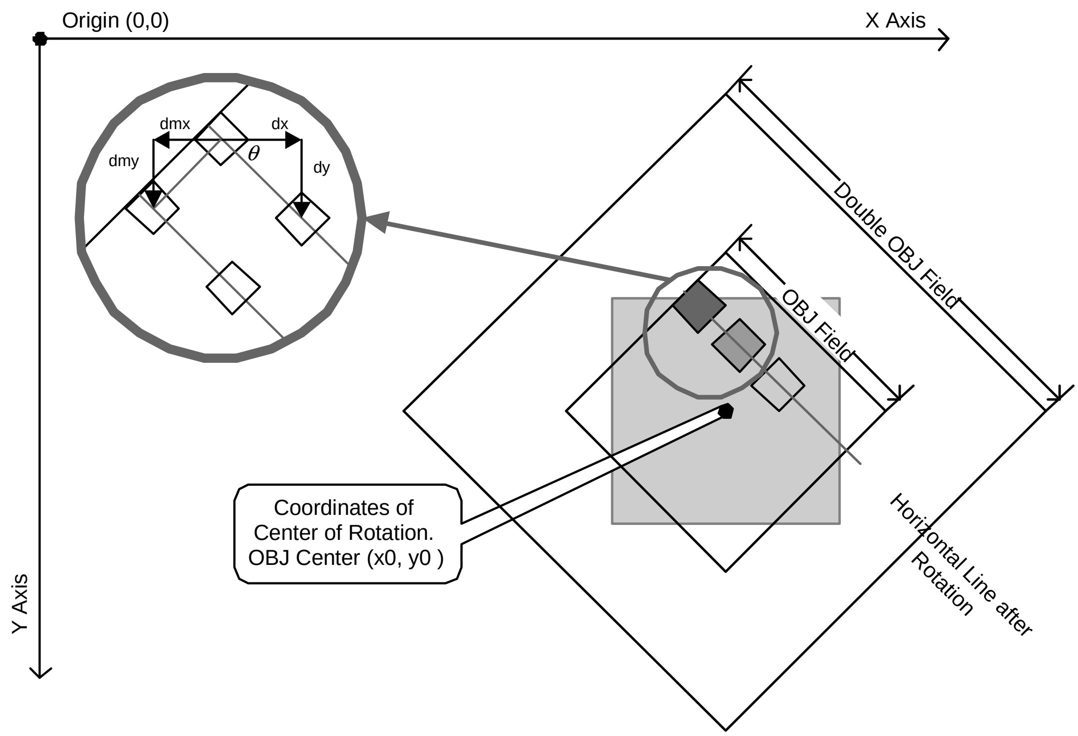
Signed fixed-point decimal (sign + 7-bit integer + 8-bit decimal part)

6.3.2.3 OBJ Rotation and Scaling (Affine Transformation)

The OBJ character data is referenced horizontally in sequence from the top left when an OBJ is displayed, so a rotated OBJ can be displayed by rotating the reference direction. The center of the rotation is fixed to the center of the OBJ field (dot boundary). If a reference point is outside the specified OBJ size, it becomes transparent.

Figure 6-22 shows the rotation and scaling process for an OBJ.

Figure 6-22 : OBJ Rotation and Scaling



dx (reference distance in x-direction for same line) = $(1/\alpha)\cos\theta$

dy (reference distance in y-direction for same line) = $-(1/\beta)\sin\theta$

dmx (reference distance in x-direction for next line) = $(1/\alpha)\sin\theta$

dmy (reference distance in y-direction for next line) = $(1/\beta)\cos\theta$

Note: α is the scale ratio along the x-axis; β is the scale ratio along the y-axis.

- OBJ Rotation and Scaling Process

  1. Affine transformation parameter numbers to be applied are specified in OBJ Attribute 1 registered in OAM. In addition, the affine transformation parameters PA, PB, PC, and PD to be applied are set in OAM using the information in Figure 6-22.
  2. The image-processing circuitry calculates the coordinates in the x-direction in relation to the data reference start point that uses the center of the OBJ field as the center of rotation by summing the cumulative increase in the x-direction (dx and dy).
  3. If the line advances, the rendering start point coordinates for the next line are calculated by summing the cumulative increase in the y-direction (dmx and dmy) in relation to the reference starting point. Then the process in Step 2 is performed.

6.3.3 Character OBJ

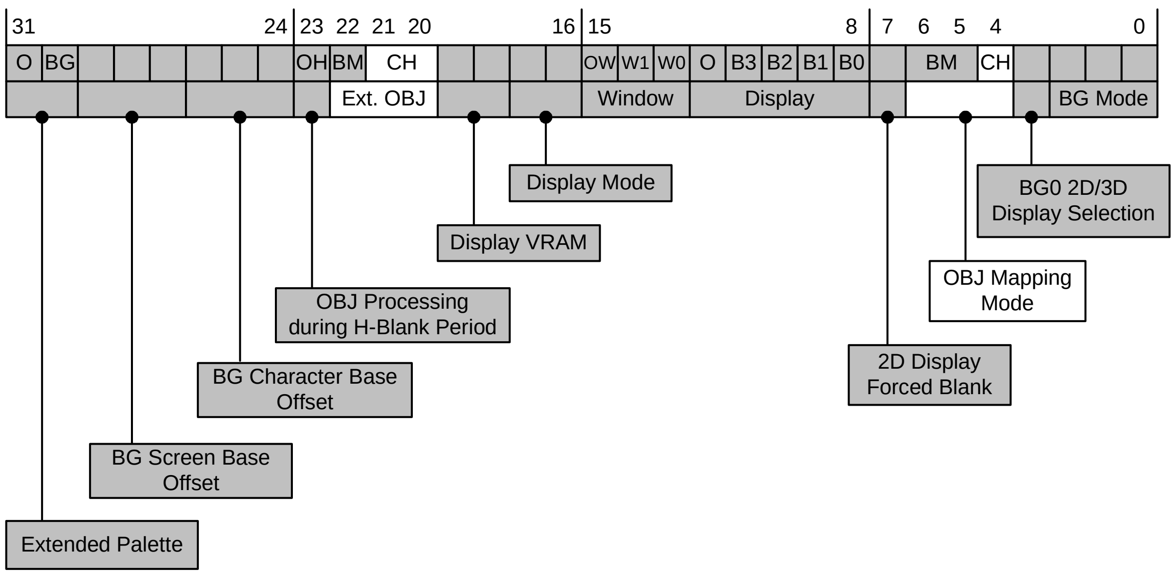
For OBJ character data, 8x8-dot sections are treated as basic characters and are assigned a Character Number. The OBJ size can be from 8x8 dots to 64x64 dots (12 different sizes). The OBJ character data base address is fixed as the VRAM base address. OBJ are defined as having either 16 colors or 256 colors, so the definition of a single basic character requires either 32 bytes or 64 bytes (both have the same format as BG character data).

The Color Mode setting in OAM OBJ Attribute 1 defines whether to reference OBJ character data in 16-color format or 256-color format. In addition, the palette specified in OBJ Attribute 2 is used when 16-color mode has been set or when 256-color mode is set when extended palettes are enabled. Extended palettes can be enabled/disabled with the DISPCNT [DB_DISPCNT] Register.

Select either 1D Mapping or 2D Mapping for character data VRAM mapping.

Display Control Register (2D Graphics Engine A)

Name: DISPCNT Address: 0x04000000 Attribute: R/W Initial Value: 0x00000000



- [d22–d20] : OBJ-VRAM Region Extended Flag
  - CH [d21–d20] : VRAM Region Extended Flag for Character OBJ

    These bits specify OBJ-VRAM capacity when OBJ character data uses 1D mapping. When set to 00, the capacity is the same as the AGB. Table 6-12 shows the starting character name boundaries that can be specified with OAM OBJ Attribute 2.

Table 6-12 : Starting Character Name Boundaries for OBJ Attribute 2

00	32 KB (Starting character name boundary: 32 bytes)
01	64 KB (Starting character name boundary: 64 bytes)
10	128 KB (Starting character name boundary: 128 bytes)
11	256 KB (Starting character name boundary: 256 bytes)

Note: When the OBJ-VRAM Region Setting flag has been set greater than the VRAM size allocated to the OBJ, do not access the region that exceeds the VRAM size allocated to the OBJ.

- [d06–d04] : OBJ Data Mapping Mode
  - CH [d04] : Character OBJ Data Mapping Mode

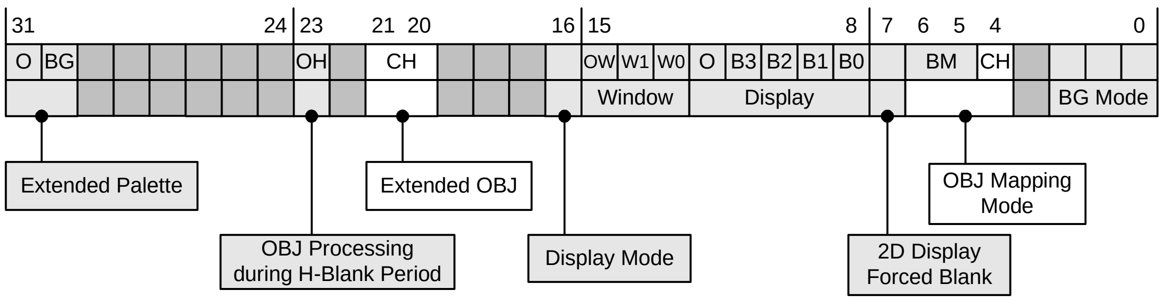
0	2D mapping
1	1D mapping

In 2D mapping mode, only up to 32 KB of OBJ-VRAM can be referenced.

In 1D mapping mode, a capacity of 32 to 256 KB can be set with the OBJ-VRAM Region Extended Flag. Accordingly, more OBJ characters can be defined in OBJ-VRAM using 1D mapping mode.

Display Control Register 1 (2D Graphics Engine B)

Name: DB_DISPCNT Address: 0x04001000 Attribute: R/W Initial Value: 0x00000000



- [d21–d20] : OBJ-VRAM Region Extended Flag
  - CH [d21–d20] : VRAM Region Extended Flag for Character OBJ

These bits specify OBJ-VRAM capacity when OBJ character data uses 1D mapping. When set to 00, the capacity is the same as the AGB. Table 6-13 shows the starting character name boundaries that can be specified with OAM OBJ Attribute 2.

Table 6-13 : Starting Character Name Boundaries for OBJ Attribute 2

00	32 KB (starting character name boundary: 32 bytes)
01	64 KB (starting character name boundary: 64 bytes)
10	128 KB (starting character name boundary: 128 bytes)
11	256 KB (starting character name boundary: 256 bytes)

Note: With 2D Graphics Engine B, the maximum size that can be allocated to VRAM is 128 KB. When the OBJ-VRAM Region Setting flag has been set greater than the VRAM size allocated to the OBJ, do not access the region that exceeds the VRAM size allocated to the OBJ.

- [d06–d04] : OBJ Data Mapping Mode
  - CH [d04] : Character OBJ Data Mapping Mode

0	2D mapping
1	1D mapping

In 2D mapping mode, only up to 32 KB of OBJ-VRAM can be referenced.

In 1D mapping mode, a capacity of 32 to 256 KB can be set with the OBJ-VRAM Region Extended Flag. Accordingly, more OBJ characters can be defined in OBJ-VRAM using 1D mapping mode.

6.3.3.1 Character Data Format

The character data format for Character OBJ is shown below. The Character Display table shows the case when an 8x8-dot character is defined.

6.3.3.1.1 16-Color Mode

The character data format for 16-color mode, correspondence between character display and pixel data, and address mapping (Figure 6-23) are shown below.

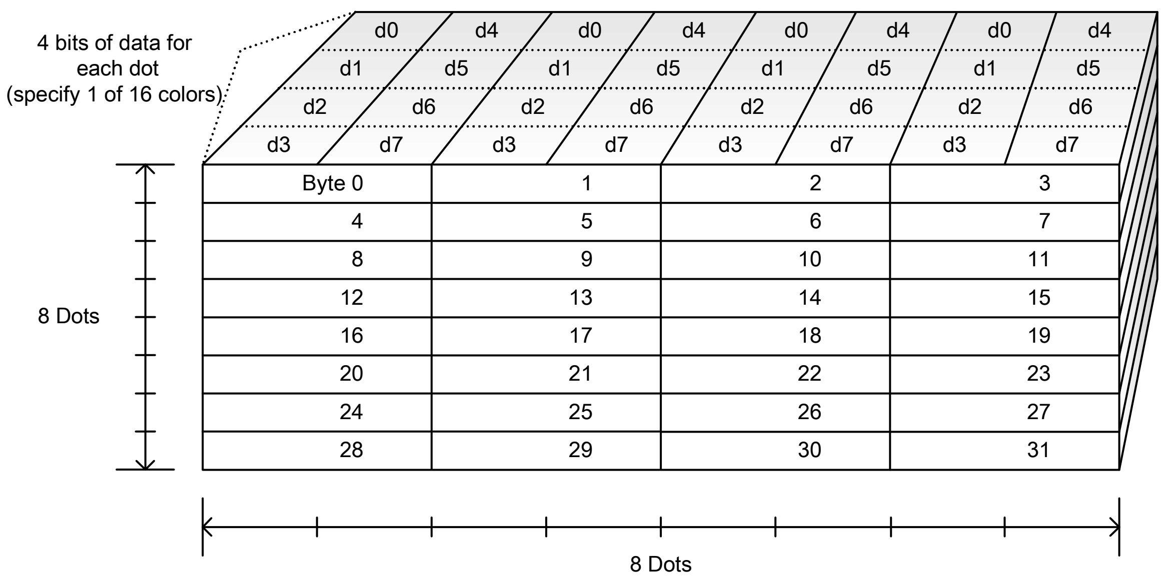
16-Color Mode Character Data

15 … 12	11 … 8	7 … 4	3 … 0
P3	P2	P1	P0
4 pixels worth of data (4 bits/pixel)			

Character Display

P0	P1	P2	P3				

Figure 6-23 : Character Data Address Mapping (16-Color Mode Character OBJ)



6.3.3.1.2 256-Color Mode

The character data format for 256-color mode, correspondence between character display and pixel data, and address mapping (Figure 6-24) are shown below.

256-Color Mode Character Data

15	12	11	8	7	4	3	0
P1				P0			
2 pixels worth of data (8 bits/pixel)							

Character Display

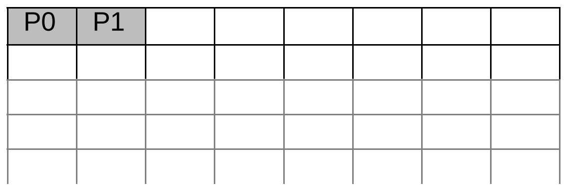
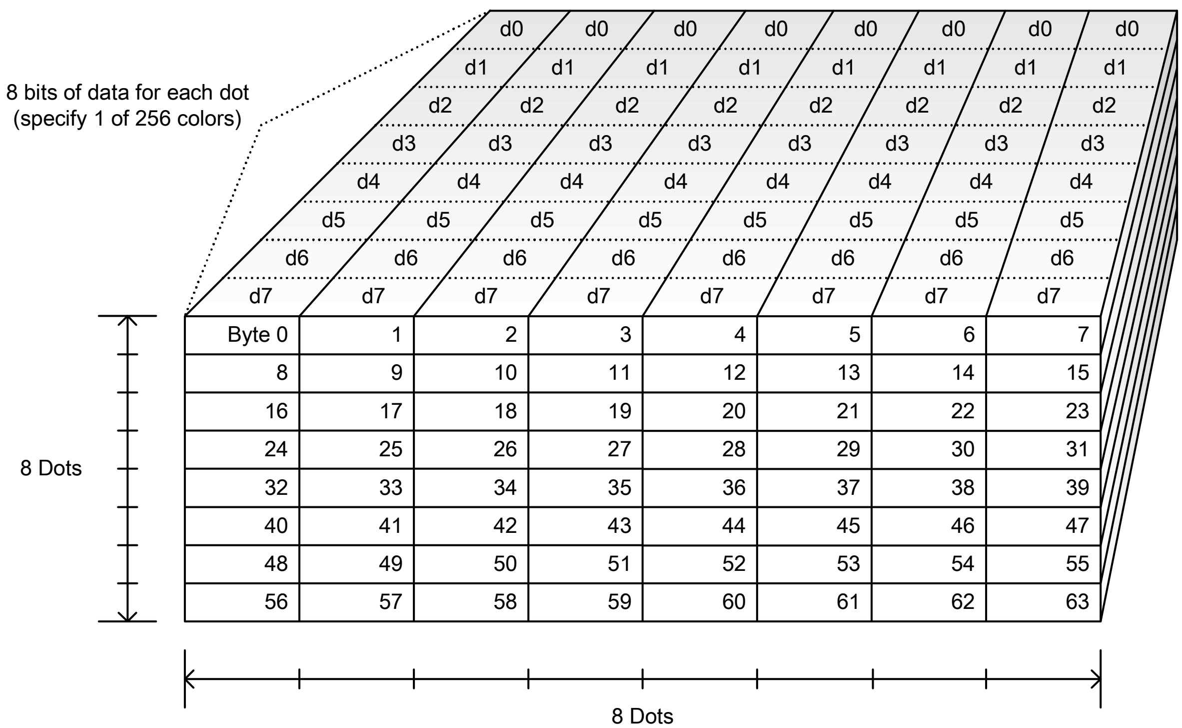


Figure 6-24 : Character Data Address Mapping (256-Color Mode Character OBJ)

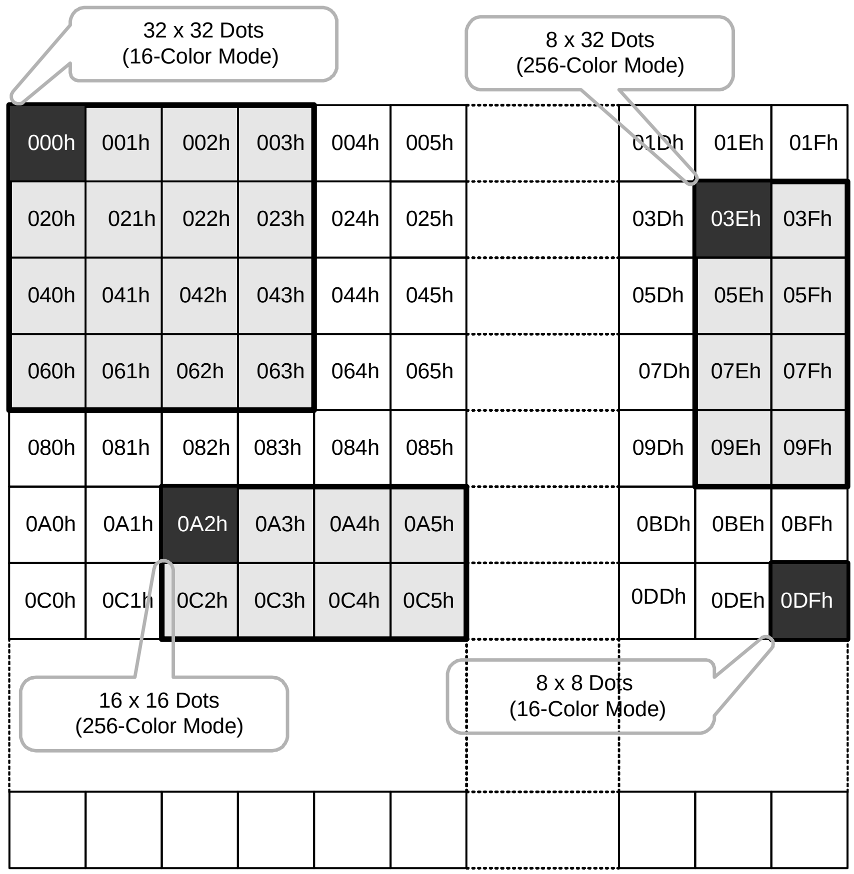


6.3.3.2 Mapping Modes for Character OBJ Data

6.3.3.2.1 2D Mapping

When displaying 256-color x 1-palette characters in 2D mapping mode, the character name specification is limited to even numbers, as shown in Figure 6-25.

Figure 6-25 : 2D Mapping



6.3.3.2.2 1D Mapping

The address where the data that makes up the character is stored is consecutive for each character, as shown in Figure 6-26 and Figure 6-27.

Figure 6-26 : 1D Mapping when Character Name Boundary is 32 Bytes

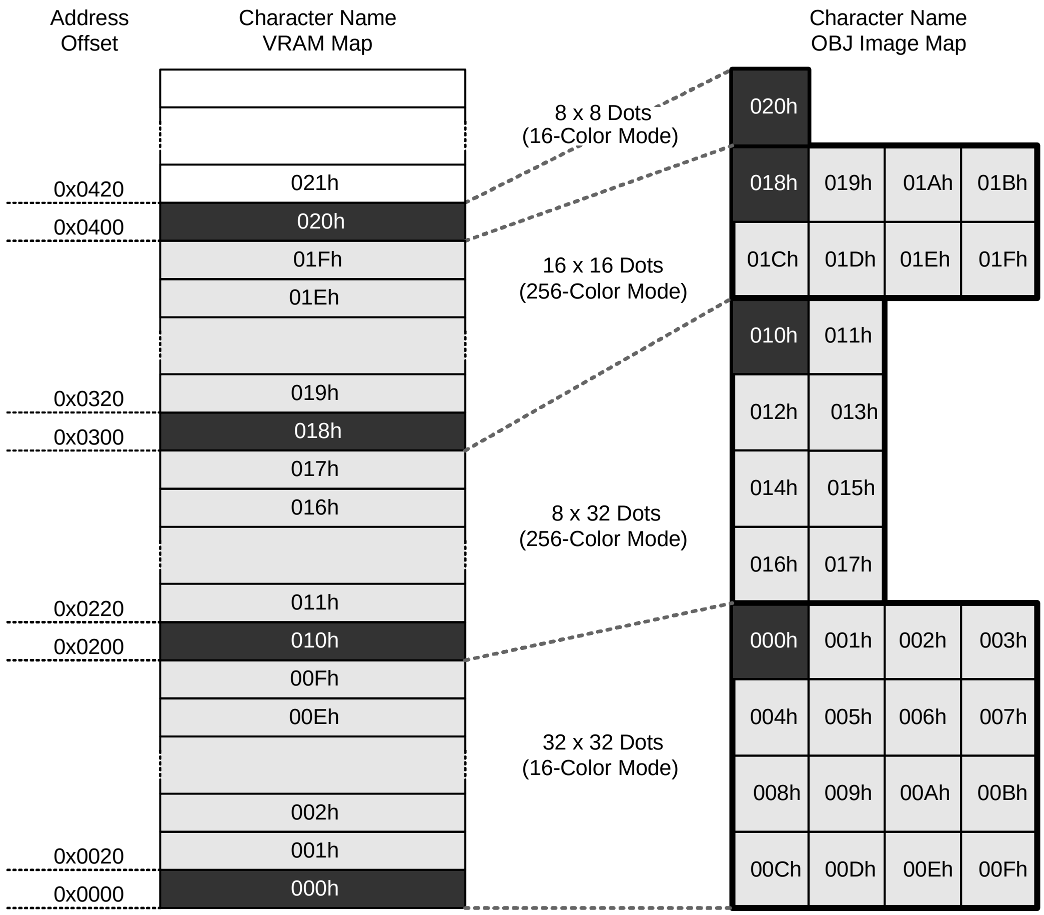
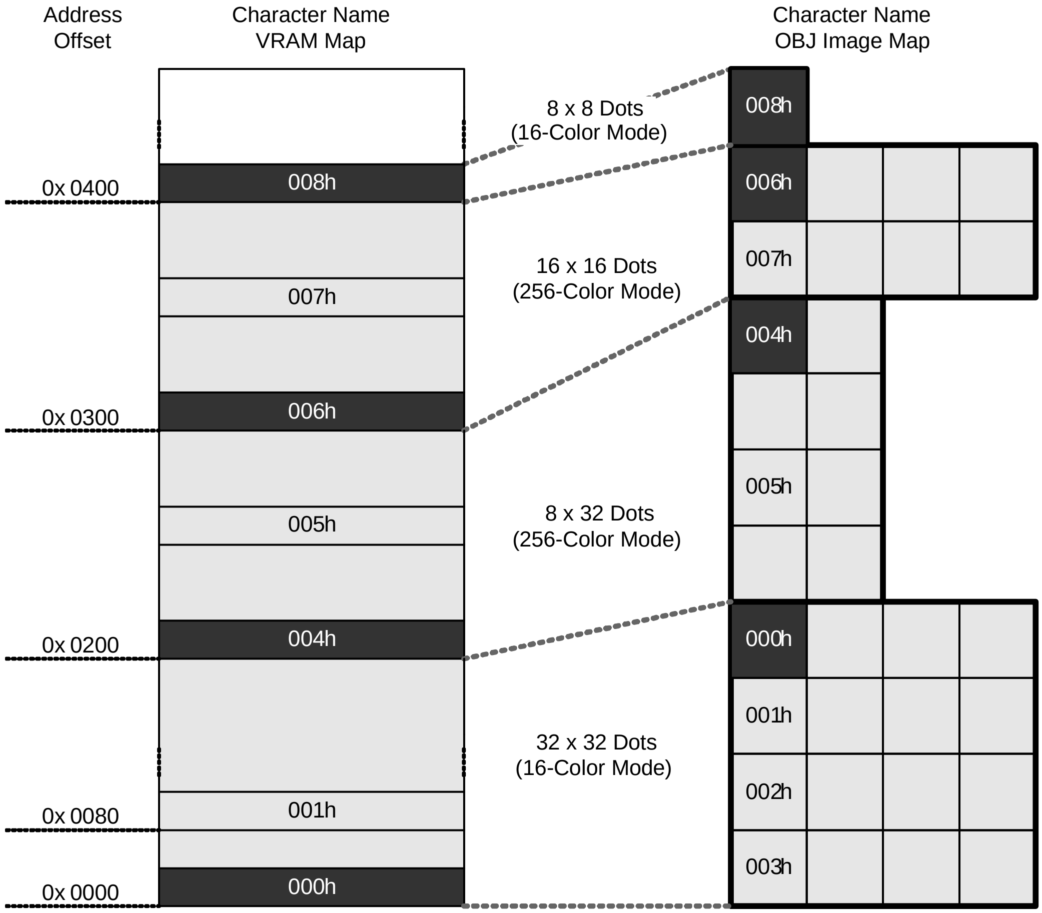


Figure 6-27 : 1D Mapping when Character Name Boundary is 128 Bytes

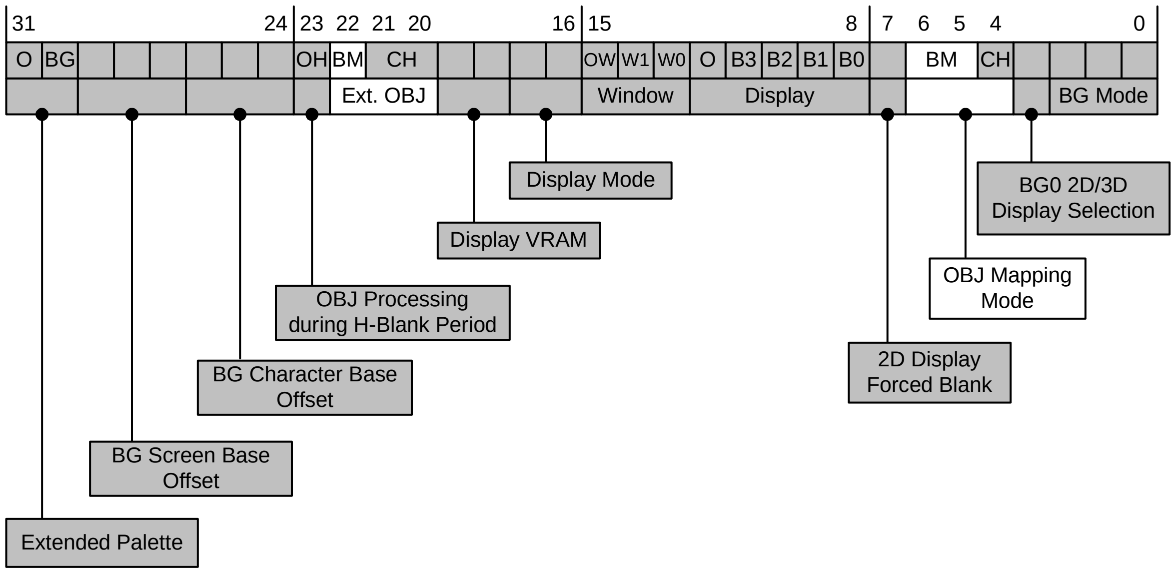


6.3.4 Bitmap OBJ

The VRAM Extended Flag and Mapping Mode for Bitmap OBJ are set with the DISPCNT [DB_DISPCNT] Register.

Display Control Register (2D Graphics Engine A)

Name: DISPCNT Address: 0x04000000 Attribute: R/W Initial Value: 0x00000000



- [d22–d20] : OBJ-VRAM Region Extended Flag
  - BM [d22] : VRAM Extended Flag for Bitmap OBJ

    These bits specify OBJ-VRAM capacity when 1D mapping is selected for OBJ bitmap data.

0	128 KB (starting character name boundary of 128 bytes)
1	256 KB (starting character name boundary of 256 bytes)

- [d06–d04] : OBJ Data Mapping Mode
  - BM [d06–d05] : Bitmap OBJ Data Mapping Mode

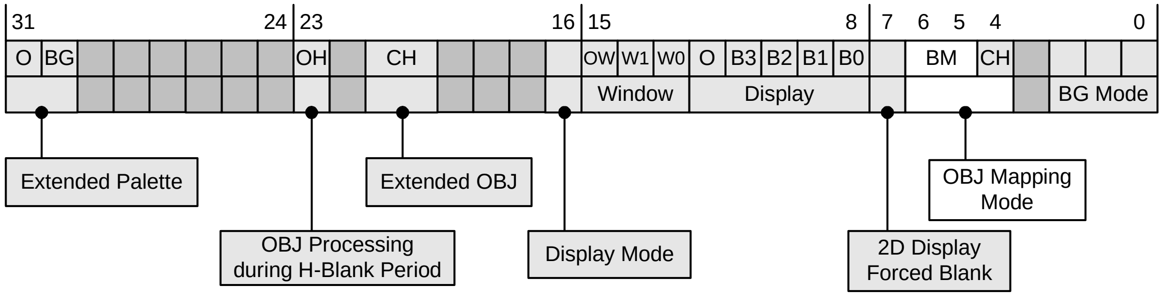
00	2D mapping with 128 horizontal dots
01	2D mapping with 256 horizontal dots
10	1D mapping
11	Prohibited Setting

The OBJ-VRAM Region Extended Flag is ignored with 2D mapping. In this case, OBJ-VRAM is referenced in a range of addresses that can be specified by the 10-bit character name set by OBJ Attribute 2 in OAM.

Capacity is set with the OBJ-VRAM Region Extended Flag with 1D mapping.

DB_DISPCNT: Display Control Register 1 (2D Graphics Engine B)

Name: DB_DISPCNT Address: 0x04001000 Attribute: R/W Initial Value: 0x00000000



- [d06–d04] : OBJ Data Mapping Mode
  - BM [d06–d05] : Bitmap OBJ Data Mapping Mode

00	2D mapping with 128 horizontal dots
01	2D mapping with 256 horizontal dots
10	1D mapping
11	Prohibited Setting

In 2D mapping mode, only up to 32 KB of OBJ-VRAM can be referenced.

In 1D mapping mode, OBJ-VRAM capacity is set to 128 KB.

Note: With 2D Graphics Engine B, a maximum of 128 KB can be allocated to VRAM. Accordingly, although it can be specified for 2D Graphics Engine A, the OBJ-VRAM capacity for 1D mapping mode is fixed to 128 KB.

6.3.4.1 Bitmap OBJ Data

Bitmap OBJ data

15	14 10	9 8 7 5	4 0
A	BLUE	GREEN	RED
α_{BMP}	P0		

- A [d15] : α_{BMP}

  The α_{BMP} value is an element of the OBJ's transparency α, where $\alpha = \alpha_{BMP} \times (\alpha_{OAM} + 1)$. Set α_{OAM} using OBJ Attribute 2 (see "6.3.2.2 OAM Data Format" on page 137).

OBJ Display

P0							

6.3.4.2 Blending with BG

As with translucent OBJ, Bitmap OBJ can be blended with the BG of the second target screen for display. When $\alpha_{OAM} = 0$, the entire region of the OBJ becomes transparent and is not rendered. When α_{OAM} is non-zero, the OBJ is blended for display according to the following formula:

$$C = \frac{C_{OBJ} \times \alpha + C_{BG} \times (16 - \alpha)}{16}$$

$\alpha = \alpha_{BMP} \times (\alpha_{OAM} + 1)$

α_{BMP} is set with Bitmap OBJ data, and α_{OAM} is a value specified with OBJ Attribute 2 of OAM.

C is the color of the blending result (calculation results are rounded to the nearest integer).

C_{OBJ} is the Bitmap OBJ color of the first target screen.

C_{BG} is the BG color of the second target screen.

TWL-06-0017-001-D
Released: February 16, 2009

6.3.4.3 Mapping Modes for Bitmap OBJ Data

6.3.4.3.1 2D Mapping with 128 Horizontal Dots

Figure 6-28 shows the 2D map of Bitmap OBJ data with 128 horizontal dots in VRAM.

Figure 6-28 : 2D Map of Bitmap OBJ Data VRAM (128 Horizontal Dots)

	Dot 0	1	2	3		125	126	127
Line 0	0h	2h	4h	6h		FAh	FCh	FEh
1	100h	102h	104h	106h		1FAh	1FCh	1FEh
2	200h	202h					2FCh	2FEh
3	300h	302h					3FCh	3FEh
4	400h							4FEh

Character names are set in units of 8x8 dots (128 bytes) of bitmap data. Figure 6-29 shows the 2D image map of character names in VRAM.

Figure 6-29 : 2D Image Map of Character Name VRAM

Dot / Line	0-7	8-15	16-23	24-31		104-111	112-119	120-127
0-7	0h	1h	2h	3h		Dh	Eh	Fh
8-15	10h	11h	12h	13h		1Dh	1Eh	1Fh
16-23	20h	21h	22h	23h		2Dh	2Eh	2Fh
24-31	30h	31h	32h	33h		3Dh	3Eh	3Fh
32-39	40h	41h	42h	43h		4Dh	4Eh	4Fh

6.3.4.3.2 2D Mapping with 256 Horizontal Dots

Figure 6-30 shows the 2D map of Bitmap OBJ data with 256 horizontal dots in VRAM.

Figure 6-30 : 2D Map of Bitmap OBJ Data VRAM (256 Horizontal Dots)

	Dot 0	1	2	3		253	254	255
Line 0	0h	2h	4h	6h		1FAh	1FCh	1FEh
1	200h	202h	204h	206h		3FAh	3FCh	3FEh
2	400h	402h					5FCh	5FEh
3	600h	602h					7FCh	7FEh
4	800h							9FEh

Character names are set in units of 8x8 dots (128 bytes) of bitmap data. Figure 6-31 shows the 2D image map of character names in VRAM.

Figure 6-31 : 2D Image Map of Character Name VRAM

Line \ Dot	0-7	8-15	16-23	24-31		232-239	240-247	248-255
0-7	0h	1h	2h	3h		1Dh	1Eh	1Fh
8-15	20h	21h	22h	23h		3Dh	3Eh	3Fh
16-23	40h	41h	42h	43h		5Dh	5Eh	5Fh
24-31	60h	61h	62h	63h		7Dh	7Eh	7Fh
32-39	80h	81h	82h	83h		9Dh	9Eh	9Fh

6.3.4.3.3 1D Mapping

Character Names

In 2D Graphics Engine A, the Bitmap OBJ VRAM extension flag in the display control register DISPCNT changes both the OBJ-VRAM range that can be specified by a character name, and the character name boundary (see Table 6-14).

In 2D Graphics Engine B, VRAM allocation is restricted, so the OBJ-VRAM range is fixed to 128 KB, and the character name boundary is fixed to 128.

Table 6-14 : Character Name Boundaries

Bitmap OBJ VRAM Extended Flag	OBJ-VRAM Specifiable Range	Character Name Boundary
0	128 KB	128 bytes
1	256 KB	256 bytes

For example, if the VRAM Extended Flag for Bitmap OBJ is set to 0, and the OBJ Attribute 2 setting for Starting Character Name is set to 4Ch, the Bitmap OBJ data defined from address 0x2600 (= 4Ch x 128 bytes) is referenced.

1D VRAM Mapping of Bitmap OBJ Data

Map bitmap data from the starting address of the character name boundary for the size of the character. This size is not in units of 8x8 dots. Figure 6-32 and Figure 6-33 show the 1D map for 8x8-dot characters and 16x16-dot characters. In these figures, `C+xxh` denotes the offset from the starting address of the character name boundary.

Figure 6-32 : 1D Map of VRAM with 8x8-Dot Characters

	0	1	2	3	4	5	6	7
0	C+0h	C+2h	C+4h	C+6h	C+8h	C+Ah	C+Ch	C+Eh
1	C+10h	C+12h	C+14h	C+16h	C+18h	C+1Ah	C+1Ch	C+1Eh
2	C+20h							C+2Eh
3	C+30h							C+3Eh
4	C+40h							C+4Eh
5	C+50h							C+5Eh
6	C+60h	C+62h	C+64h	C+66h	C+68h	C+6Ah	C+6Ch	C+6Eh
7	C+70h	C+72h	C+74h	C+76h	C+78h	C+7Ah	C+7Ch	C+7Eh

Figure 6-33 : 1D Map of VRAM with 16x16-Dot Characters

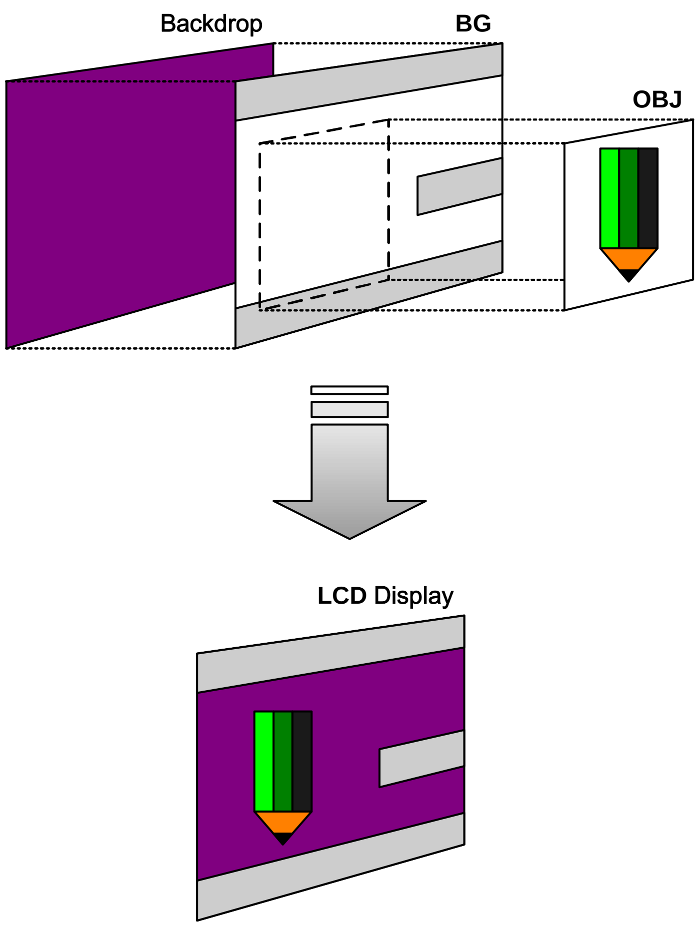
	0	1	2		13	14	15
0	C+0h	C+2h	C+4h		C+1Ah	C+1Ch	C+1Eh
1	C+20h	C+22h	C+24h		C+3Ah	C+3Ch	C+3Eh
2	C+40h						C+5Eh
13	C+1A0h						C+1BEh
14	C+1C0h	C+1C2h	C+1C4h		C+1DAh	C+1DCh	C+1DEh
15	C+1E0h	C+1E2h	C+1E4h		C+1FAh	C+1FCh	C+1FEh

6.4 Backdrop

The 2D graphics displayed on the LCD are composed of OBJ, BG, and the Backdrop. An OBJ is a relatively small image, but several of them can be displayed. They are mainly used to display characters that move around the screen. A BG has features equivalent to an OBJ, but only a few BG screens can be displayed because a BG is large and consumes a lot of memory. A BG is used to display large images such as objects that are continuously on-screen or in the background.

On TWL, regions of the LCD screen where no OBJ and BG are displayed are filled with a single color. This region is called the *Backdrop* and can be visualized as a single-color surface that is always displayed furthest in the back, as depicted in Figure 6-34. The Backdrop is a surface filled only with a single color and does not have the features of OBJ and BG. The Backdrop color can be changed with the palette (see "6.5 Color Palettes" on page 159).

Figure 6-34 : Backdrop Schematic



6.5 Color Palettes

As a standard feature, TWL has RAM allocated specifically for BG and OBJ palettes (Palette RAM). Data stored in Palette RAM are called standard palettes.

A BG or OBJ can be displayed using just a standard palette, but extended palettes allow the use of 256 colors x 16 palettes and enable richer visuals. To use extended palettes, allocate VRAM using the RAM Control Register and enable the Extended Palette Flag with the DISPCNT [DB_DISPCNT] Register.

6.5.1 Standard Palettes

Standard palette RAM is allocated separately for OBJ and for BG in both 2D Graphics Engine A and 2D Graphics Engine B. Color 0 in each palette is the transparent color, regardless of the settings. The Backdrop screen uses the color set at the beginning of the BG palette (Color 0 of Palette 0). Because standard palette RAM resides inside the 2D Graphic Engines, the 2D Graphic Engine must be enabled in the Power Control Register (POWCNT) before data can be written to its RAM.

Figure 6-35 shows the standard palette RAM addresses. Figure 6-36 shows the color specifications for 16 Colors x 16 Palettes. Figure 6-37 shows the color specifications for 256 Colors x 1 Palette.

Figure 6-35 : Standard Palette RAM Addresses (Add +0x400 for 2D Graphics Engine B)

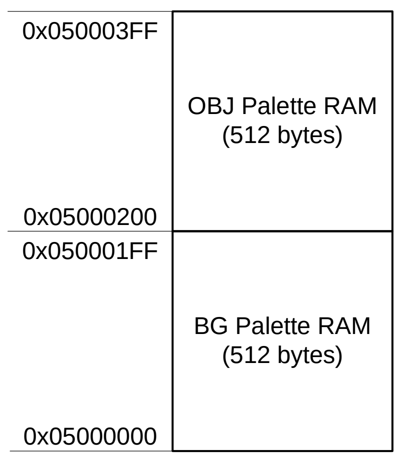


Figure 6-36 : 16 Colors x 16 Palettes

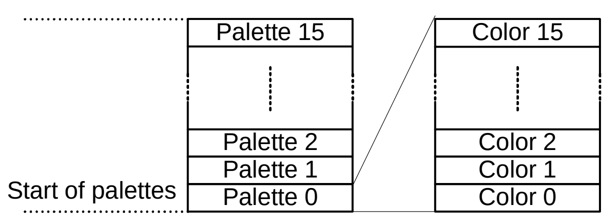
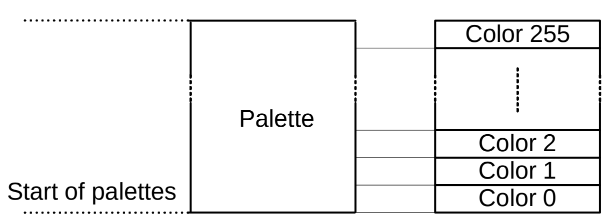


Figure 6-37 : 256 Colors x 1 Palette



The format for color data is shown below.

Color Data Format

15	14 – 10	9 – 5	4 – 0
GREEN	BLUE	GREEN	RED
	Color Data		

Note: The 15th bit of the color data is used as the lowest-order green bit. The lowest-order blue and red bits are extended with a 0, and the RGB values are calculated using 6 bits each.

	Color Data

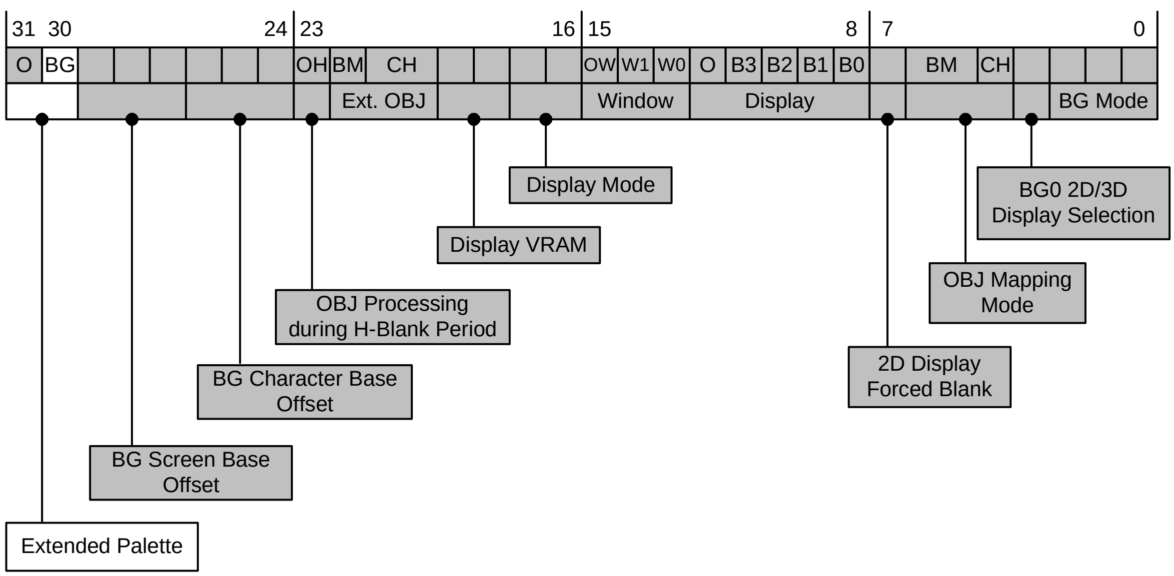
6.5.2 Extended Palettes

OBJ and each BG screen can be allocated 256 colors x 16 palettes (8 KB) of VRAM by setting the Extended Palette Flag in the DISPCNT [DB_DISPCNT] Register and the RAM Bank Control Register. When allocated, palette slots are not mapped to the CPU bus. To rewrite the palette data, the palette slot must be allocated to LCDC.

6.5.2.1 BG Extended Palettes

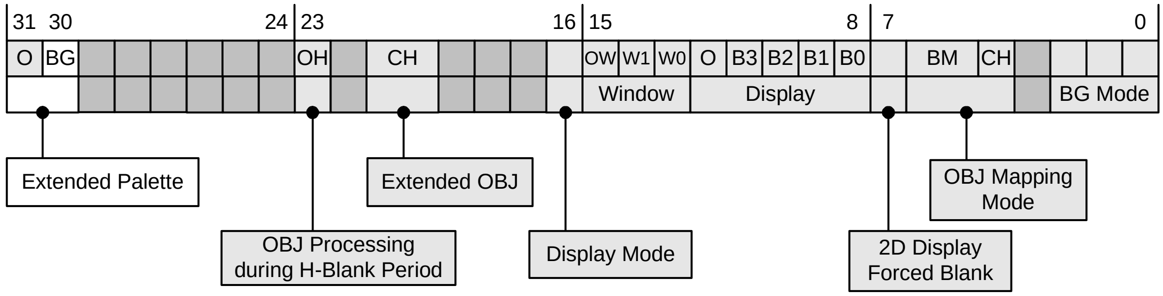
Display Control Register (2D Graphics Engine A)

Name: DISPCNT Address: 0x04000000 Attribute: R/W Initial Value: 0x00000000



Display Control Register 1 (2D Graphics Engine B)

Name: DB_DISPCNT Address: 0x04001000 Attribute: R/W Initial Value: 0x00000000



- [d31,d30] : Extended Palette Enable Flag
  - BG [d30] : BG Extended Palette

    This flag is valid for BG screens that can be displayed with 256 Colors x 16 Palettes.

0	Disable (256 colors x 1 palette)
1	Enable (256 colors x 16 palettes)

    The standard palette is always used for BG screens that do not support 256 Colors x 16 Palettes, even if BG Extended Palettes are enabled. In addition, the Backdrop screen always uses Color 0 of the standard palette.

    To use BG extended palettes, VRAM must be allocated to the BG Extended Palette Slots. See the RAM Bank Control Register for allocating VRAM to the BG Extended Palette Slots.

BG Extended Palette Slots

BG Extended Palettes can have up to 32 KB allocated to Slots 0-3. Whether BG0 uses Slot 0 or Slot 2 is selected with the BG0 Control Register, and whether BG1 uses Slot 1 or Slot 3 is selected with the BG1 Control Register. BG2 can use only Slot 2, and BG3 can use only Slot 3. Therefore, if Slot 0 is set to BG0 and Slot 1 to BG1, each BG screen can use its own extended palette. On the other hand, by setting Slots 2 and 3 to be shared by all BG screens, the BG Extended Palettes can conserve 16 KB.

Color 0 in both palettes is the transparent color, regardless of the settings. The Backdrop screen uses the color set at the beginning of the BG standard palette (Color 0 of Palette 0).

Figure 6-38 shows the memory map for BG extended palette slots.

Figure 6-38 : BG Extended Palette Memory Map

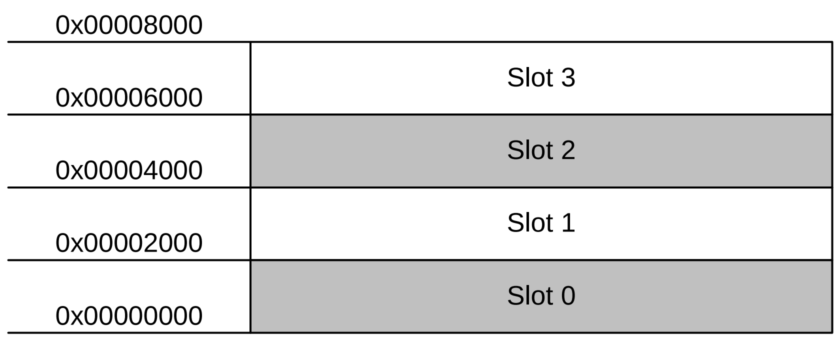


Table 6-15 lists the palettes that can be used by each type of BG.

Table 6-15 : Palettes and BG Types

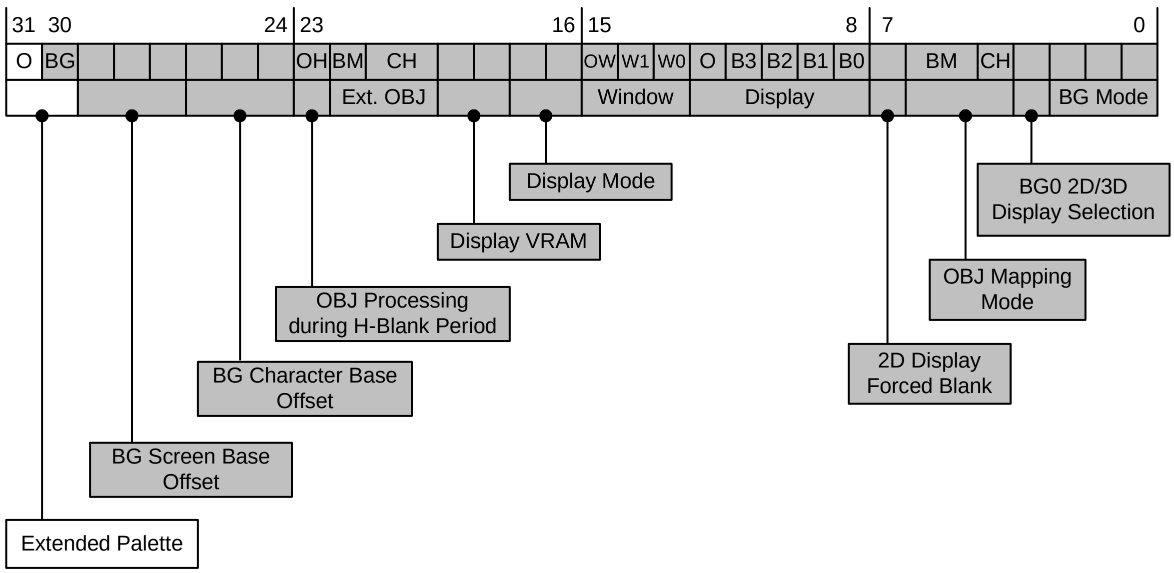
Category	BG Type		Colors/ Palette	BG Screen	Usable Palette Region				
					Standard	Extended Palette Slot			
						0	1	2	3
Character BG	Text BG		16/16	BG0–3	X				
			256/16	BG0	X	X		X	
				BG1	X		X		X
				BG2	X			X	
				BG3	X				X
	Affine BG		256/1	BG2–3	X				
	Ext. BG	256-Color x 16-Palette BG	256/16	BG2	X			X	
				BG3	X				X
Bitmap BG		256-Color	256/1	BG2–3	X				
		Direct Color	32,768	BG2–3					
	Large-Screen 256-Color		256/1	BG2	X				

As shown in Figure 6-15, the Extended Palette Slot number can be selected for BG0 and for BG1. With 2D Graphics Engine B, Large-Screen 256-Color Bitmap BG cannot be selected for the BG type. See the section on the BG Control Register in "6.2.2 BG Control" on page 105 to learn how to select slots.

6.5.2.2 OBJ Extended Palettes

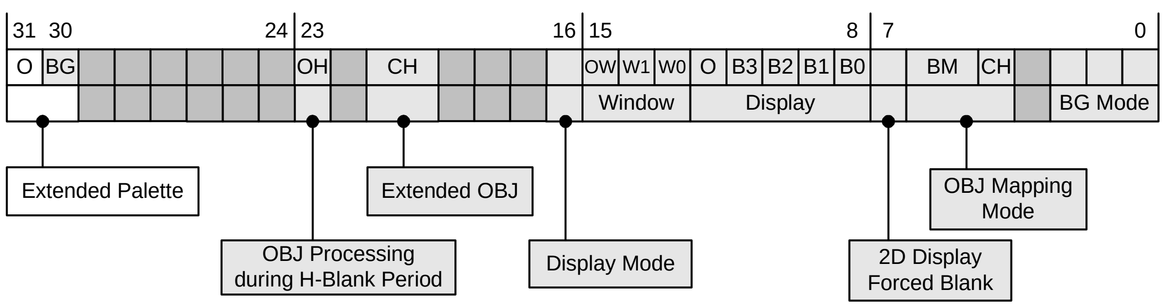
Display Control Register (2D Graphics Engine A)

Name: DISPCNT Address: 0x04000000 Attribute: R/W Initial Value: 0x00000000



Display Control Register 1 (2D Graphics Engine B)

Name: DB_DISPCNT Address: 0x04001000 Attribute: R/W Initial Value: 0x00000000



- [d31,d30] : Extended Palette Enable Flag
  - O [d31] : OBJ Extended Palette

0	Disable (256 colors x 1 palette)
1	Enable (256 colors x 16 palettes)

The standard palette (palette RAM) is always used for 6-Color Mode OBJ, even if OBJ Extended Palettes are enabled. See "6.5 Color Palettes" on page 159 for more information on these palettes.

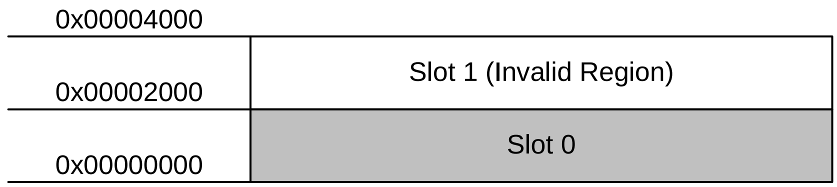
To use OBJ extended palettes, VRAM must be allocated to the OBJ Extended Palette Slots. See the RAM Bank Control Register for allocating VRAM to the OBJ Extended Palette Slots.

OBJ Extended Palette Slots

Although 16 KB of VRAM is allocated to the OBJ extended palettes, only 8 KB of this can be used as an extended palette. As Figure 6-39 illustrates, only Slot 0 can be used as an extended palette; Slot 1 is invalid.

Color 0 for each palette is handled as the transparent color, regardless of the settings. The Backdrop screen uses the color set at the beginning of the standard BG palette (Color 0 of Palette 0).

Figure 6-39 : OBJ Extended Palette Slot Memory Map



6.6 Windows

Window features can restrict the regions where BG and OBJ screens are displayed as well as the region where color special effects are applied. TWL uses three kinds of windows: Window 0, Window 1, and the OBJ Window (see OBJ Attribute 0 in "6.3 OBJ" on page 133 to read about the OBJ Window settings).

Window Interior Control Register

Name	Address	Attribute	Initial Value
(2D_A) WININ	0x04000048	R/W	0x0000
(2D_B) DB_WININ	0x04001048	R/W	0x0000

15		13	12				8	7		5	4				0
		EFCT	OBJ	BG3	BG2	BG1	BG0			EFCT	OBJ	BG3	BG2	BG1	BG0
Inside Window 1								Inside Window 0							

Window Exterior and OBJ Window Interior Control Register

Name	Address	Attribute	Initial Value
(2D_A) WINOUT	0x0400004A	R/W	0x0000
(2D_B) DB_WINOUT	0x0400104A	R/W	0x0000

15		13	12				8	7		5	4				0
		EFCT	OBJ	BG3	BG2	BG1	BG0			EFCT	OBJ	BG3	BG2	BG1	BG0
Inside OBJ Window								Outside Window (0,1, and OBJ Windows)							

- [d13], [d05] : EFCT: Color Special Effects Enable Flag

0	Disable
1	Enable

- OBJ, BG3-0 [d12–d08], [d04–d00] : Display Enable Flag

0	Hide
1	Show

Display control over the window's exterior is valid whenever Window 0, Window 1, or the OBJ Window is displayed.

Window Position Setting Register

Name	Address	Attribute	Initial Value
(2D_A) WINxH(x=0, 1)	0x04000040, 0x04000042	W	0x0000
(2D_B) DB_WINxH(x=0, 1)	0x04001040, 0x04001042	W	0x0000

15	8	7	0
Window Top Left x-coordinate		Window Bottom Right x-coordinate	

- [d15–d08] : Window Top-Left x-coordinate

  Set this in the range between 0 and 255.
- [d07–d00] : Window Bottom-Right x-coordinate

  Set this in the range between 0 and 255.

Name	Address	Attribute	Initial Value
(2D_A) WINxV (x=0, 1)	0x04000044, 0x04000046	W	0x0000
(2D_B) DB_WINxV (x=0, 1)	0x04001044, 0x04001046	W	0x0000

15	8	7	0
Window Top Left y-coordinate		Window Bottom Right y-coordinate	

- [d15–d08] : Window Top-Left y-coordinate

  Set this in the range between 0 and 191.
- [d07–d00] : Window Bottom-Right y-coordinate

  Set this in the range between 0 and 192.

Window Range

If the window's top-left coordinates are (lx, ly) and the bottom-right coordinates are (rx, ry), the window range for LCD coordinates (0, 0) – (255, 191) is (lx, ly) – (rx-1, ry-1).

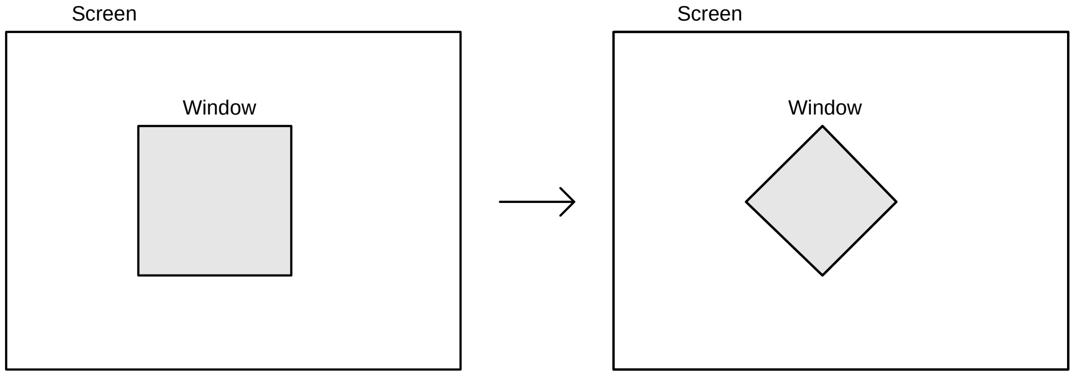
To locate a window along the right side of the LCD screen, set its bottom-right x-coordinate to 0. To locate a window along the left side of the LCD screen, set the top-left x-coordinate to 0. However, if both x-coordinates are set to 0, the window is not displayed. Consequently, the window cannot span the entire LCD screen width. Use another window (or the OBJ Window) to span the entire width of the LCD screen with windows.

Note: If the circuit revisions to the 2D graphics engine are enabled in the system configuration in TWL mode, the window width can be made to span the entire width of the LCD by setting x-coordinates of both the upper left and the lower-right coordinates to 0. (This is set using the SCFG_EXT register.)

Window Shape

Window 0 and Window 1 can be set only as rectangular shapes. However, the shape's appearance can be altered by overwriting the Window Position Setting Register during an H-Blank period (see Figure 6-40).

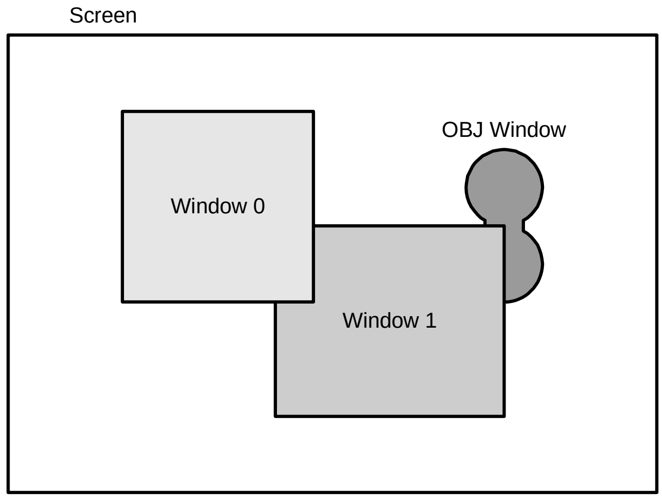
Figure 6-40 : Altering a Window Shape



6.6.1 Precedence of Windows

As shown in Figure 6-41, Window 0 always has display priority (precedence) over Window 1, and the OBJ Window has the lowest precedence. The precedence cannot be changed.

Figure 6-41 : Display Priority of Window 0, Window 1, and the OBJ Window



Note 1 Regarding Windows: When the top-left y-coordinate of the window is between 0 and 6, the top-left y-coordinate is forcibly displayed as though it were 0. Use one of the following two methods to work around this problem.

Note: If the circuit revisions to the 2D graphics engine are enabled in the system configuration in TWL mode, this problem will be avoided. (This is set using the SCFG_EXT register.)

Method 1: Perform the following steps:

1. Set the window's y-coordinate to a value of 7 or higher before the V-Count reaches 256 during the V-Blank process.
2. Restore the window's y-coordinate to its original value after confirming that V-Count has reached 262 by checking V-Count using a V-Count Match Interrupt or an H-Blank Interrupt.

Method 2: Perform the following steps:

1. Set the Window Display Enable Flag in the Display Control Register, using the values shown below after confirming that V-Count has reached 262 by checking V-Count using a V-Count Match Interrupt or an H-Blank Interrupt.

When the window y-coordinate is 0:	1 (Show window)
When the window y-coordinate is non-zero:	0 (Hide window)

2. Read the V-Count value using an H-Blank Interrupt and compare that value to (window y-coordinate – 1). If the two values are equal, set the Window Display Enable Flag in the Display Control Register to 1 (Show window) during the H-Blank.

Note 2 Regarding Windows: Immediately after drawing a line in which the bottom-right x coordinate is set to 0, drawing of the current window will not be complete at H-Blank. If the coordinates of the next window are set in this state, the window will continue drawing up to the bottom-right x coordinate of the next window.

To avoid this, you must wait until the window has finished drawing before setting the coordinates of the next window. If the bottom-right x coordinate is 0, the window will finish drawing 3 clock cycles of the system clock (33 MHz) after the H-Blank flag changes from 1 to 0 (or the V-Count value increments by 1). Between this time and the time when the H-Count reaches the top-left x coordi-

nate of the next window, it will be outside the window. So set the coordinates for the next window during this period. Note that you cannot use this method when setting window coordinates using H-Blank–initiated DMA.

6.7 Color Special Effects

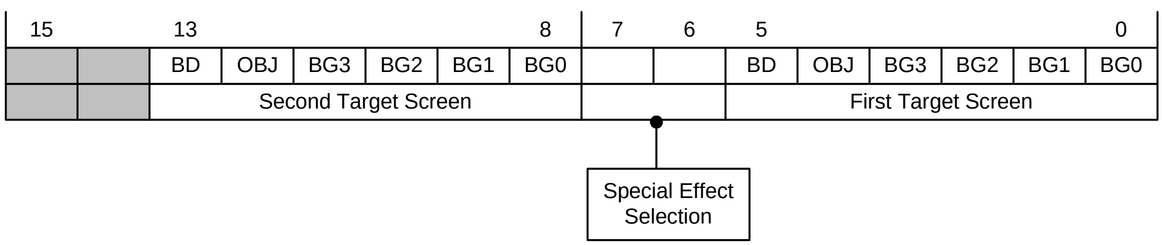
OBJ and BG can use the alpha-blending and fade-in/fade-out color special effects. These effects can be limited to a region by using windows (see "6.6 Windows" on page 166). Table 6-16 summarizes color special effects.

Table 6-16 : Color Special Effects

Color Special Effect	Result
Alpha-Blending	Computations are conducted and a 16-level translucency process is performed on two selected screens. This process is not performed on transparent portions (transparent pixels).
Brightness Up/Down (Fade-in/Fade-out)	Computations are conducted and a 16-level process of changing the brightness is performed on the selected screen. This process is not performed on transparent portions (transparent pixels).

Color Special Effect Control Register

Name	Address	Attribute	Initial Value
(2D_A) BLDCNT	0x04000050	R/W	0x0000
(2D_B) DB_BLDCNT	0x04001050	R/W	0x0000



Color special effects are set with the BLDCNT [DB_BLDCNT] Register. For alpha-blending, which processes two screens, the two target screens must have the proper priority. In addition, translucent OBJ are specified separately in OAM and the BLDCNT [DB_BLDCNT] Register specifies color special effects for the entire OBJ.

- [d07–d06] : Special Effect Selection

Table 6-17 : Color Special Effects and Processing

Effect Selection		Type	Description of the Color Special Effect Processing
d07	**d06**		
0	**0**	**None**	Normally, color special effect processing is not performed. However, 16-level translucency processing (alpha-blending) is performed only if a second target screen is directly behind a translucent OBJ, Bitmap OBJ, or a 3D surface is rendered to a BG0 screen.
0	**1**	**Alpha-Blending**	16-level translucency processing (alpha-blending) is performed only if a second target screen is directly behind the first target screen. Set the first target screen's Backdrop screen bit to off ([d05] = 0). If OBJ = 1 for the first target screen, processing is executed on all OBJ, regardless of type. If OBJ = 0, processing is executed only for translucent OBJ, Bitmap OBJ, or a 3D surface rendered to a BG0 screen.
1	**0**	**Brightness Up (see note)**	Gradually increases the brightness of the first target screen. If the first target screen specifies OBJ = 1, processing is executed only for normal OBJ. alpha-blending is always performed when a second target screen is directly behind a translucent OBJ, Bitmap OBJ, or 3D surface rendered to a BG0 screen.
1	**1**	**Brightness Down (see note)**	Gradually decreases the brightness of the first target screen. If the first target screen specifies OBJ = 1, processing is executed only for normal OBJ. alpha-blending is always performed when a second target screen is directly behind a translucent OBJ, Bitmap OBJ, or 3D surface rendered to a BG0 screen.

Note: As stated in Table 6-17, alpha-blending is always performed on translucent OBJ, Bitmap OBJ, and BG0 rendered with a 3D surface and a second target screen, where the second target screen is directly behind, regardless of the Special Effect Selection setting. Therefore, to use Fade-in/Fade-out with these screens, do not specify a second target screen (clear all) or place something other than a second target screen immediately behind these screens.

TWL-06-0017-001-D
Released: February 16, 2009

1. Alpha-Blending

Color Special Effect Alpha-Blending Factor Register

Name	Address	Attribute	Initial Value
(2D_A) BLDALPHA	0x04000052	R/W	0x0000
(2D_B) DB_BLDALPHA	0x04001052	R/W	0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
			EVB								EVA				

The factors used for alpha-blending are set with EVA and EVB in the BLDALPHA [DB_BLDALPHA] Register. EVA and EVB are divided by 16 and are used as the pixel color factors in the equations below (when EVA or EVB exceeds 16, it is reset to 16).

Note that when a Bitmap OBJ is blended with the second target screen, the Bitmap OBJ's alpha value is used instead of these values. For further details, see "6.3.4 Bitmap OBJ" on page 151 and "6.3.4.2 Blending with BG" on page 153.

- Computations for alpha-blending (16 levels of translucency)

  Display color (R) = 1st pixel color (R) x (EVA /16) + 2nd pixel color (R) x (EVB / 16)

  Display color (G) = 1st pixel color (G) x (EVA /16) + 2nd pixel color (G) x (EVB / 16)

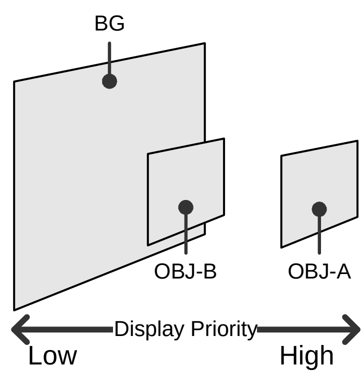
  Display color (B) = 1st pixel color (B) x (EVA /16) + 2nd pixel color (B) x (EVB / 16)

The computation results for alpha-blending are rounded to the nearest integer.

Note: An OBJ cannot be alpha-blended with another OBJ.

Figure 6-42 shows a case where an OBJ is specified as the first target screen, and a BG and an OBJ are specified as the second target screen. In this situation, OBJ-B is ignored as the target pixels for alpha-blending, and alpha-blending is carried out with OBJ-A and BG, just as if the BG were located directly behind OBJ-A.

Figure 6-42 : Alpha-Blending Display Priority



2. Brightness Up/Down

Color Special Effect Brightness Change Factor Register

Name	Address	Attribute	Initial Value
(2D_A) BLDY	0x04000054	W	0x0000
(2D_B) DB_BLDY	0x04001054	W	0x0000

15							8	7							0
											EVY				

The factor used for changing brightness is set with EVY in the BLDY [DB_BLDY] Register. EVY is divided by 16 and is used as the pixel color factor in the equations below (when EVY exceeds 16, it is reset to 16.)

- Computations to increase brightness

  Display color (R) = 1st pixel (R) + (31 - 1st pixel (R)) x (EVY / 16)

  Display color (G) = 1st pixel (G) + (31 - 1st pixel (G)) x (EVY / 16)

  Display color (B) = 1st pixel (B) + (31 - 1st pixel (B)) x (EVY / 16)

- Computations to decrease brightness

  Display color (R) = 1st pixel (R) - 1st pixel (R) x (EVY / 16)

  Display color (G) = 1st pixel (G) - 1st pixel (G) x (EVY / 16)

  Display color (B) = 1st pixel (B) - 1st pixel (B) x (EVY / 16)

The computation results for brightness are rounded to the nearest integer.

6.8 Mosaic

The Mosaic size is set with the MOSAIC [DB_MOSAIC] Register. Mosaic is turned on/off for each BG with the Mosaic Flag on the BG Control Register.

Mosaic Register

Name	Address	Attribute	Initial Value
(2D_A) MOSAIC	0x0400004C	W	0x0000
(2D_B) DB_MOSAIC	0x0400104C	W	0x0000

15	8	7	0
V Size	H Size	V Size	H Size
OBJ Mosaic Size		BG Mosaic Size	

The Mosaic Size value specifies how many dots of a normal display should comprise each large dot displayed. The Mosaic display starts with the top-left dot on the screen and uses the dots spaced a distance of the Mosaic size from the top-left dot. All other dots are overwritten with the mosaic (see Figure 6-43). In other words, if the Mosaic size is set to 0, images display normally, even if Mosaic is on.

Figure 6-43 : Display Changes According to Mosaic Size

Normal Display

00	01	02	03	04	05	06	07
10	11	12	13	14	15	16	17
20	21	22	23	24	25	26	27
30	31	32	33	34	35	36	37
40	41	42	43	44	45	46	47
50	51	52	53	54	55	56	57
60	61	62	63	64	65	66	67
70	71	72	73	74	75	76	77

Mosaic H Size: 1
V Size: 1

00	00	02		04		06	
00	00						
20		22		24		26	
40		42		44		46	
60		62		64		66	

Mosaic H Size:3
V Size: 5

00	00	00	00	04			
00	00	00	00				
00	00	00	00				
00	00	00	00				
00	00	00	00				
00	00	00	00				
60				64			

6.9 Display Priority

- BG Display Priority

  Four levels of display priority can be set for BGs with the BG Control Register. When BGs have the same priority, the one with the lower BG number has higher priority. The Backdrop screen always has the lowest priority.

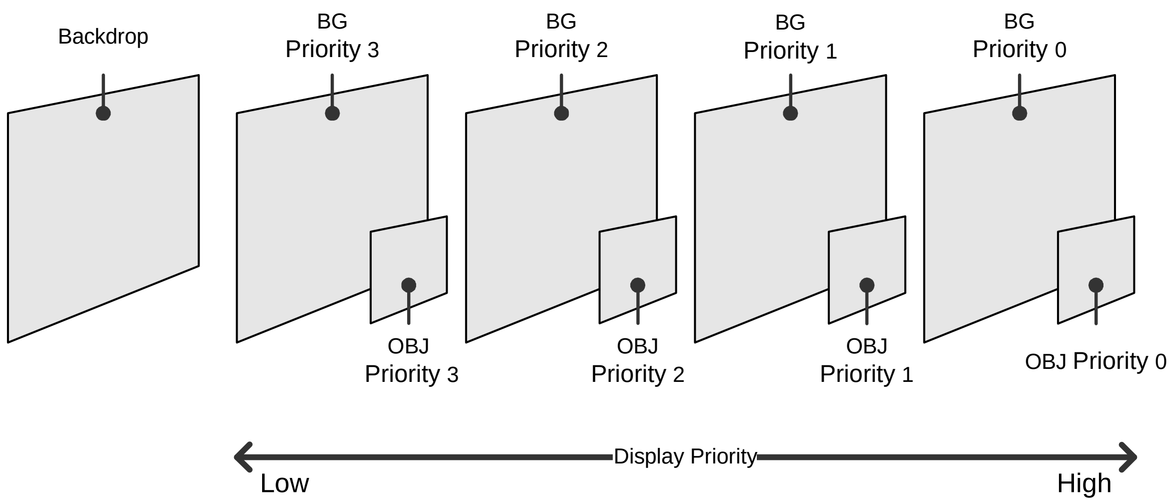
- OBJ Display Priority

  Four levels of display priority can be set for OBJs with the OBJ Attribute Data stored in OAM. When OBJs have the same priority, the one with the lower OBJ number has higher priority.

- BG - OBJ Display Priority

  If an OBJ and a BG have the same priority, the OBJ has higher priority than the BG (see Figure 6-44).

Figure 6-44 : Display Priority

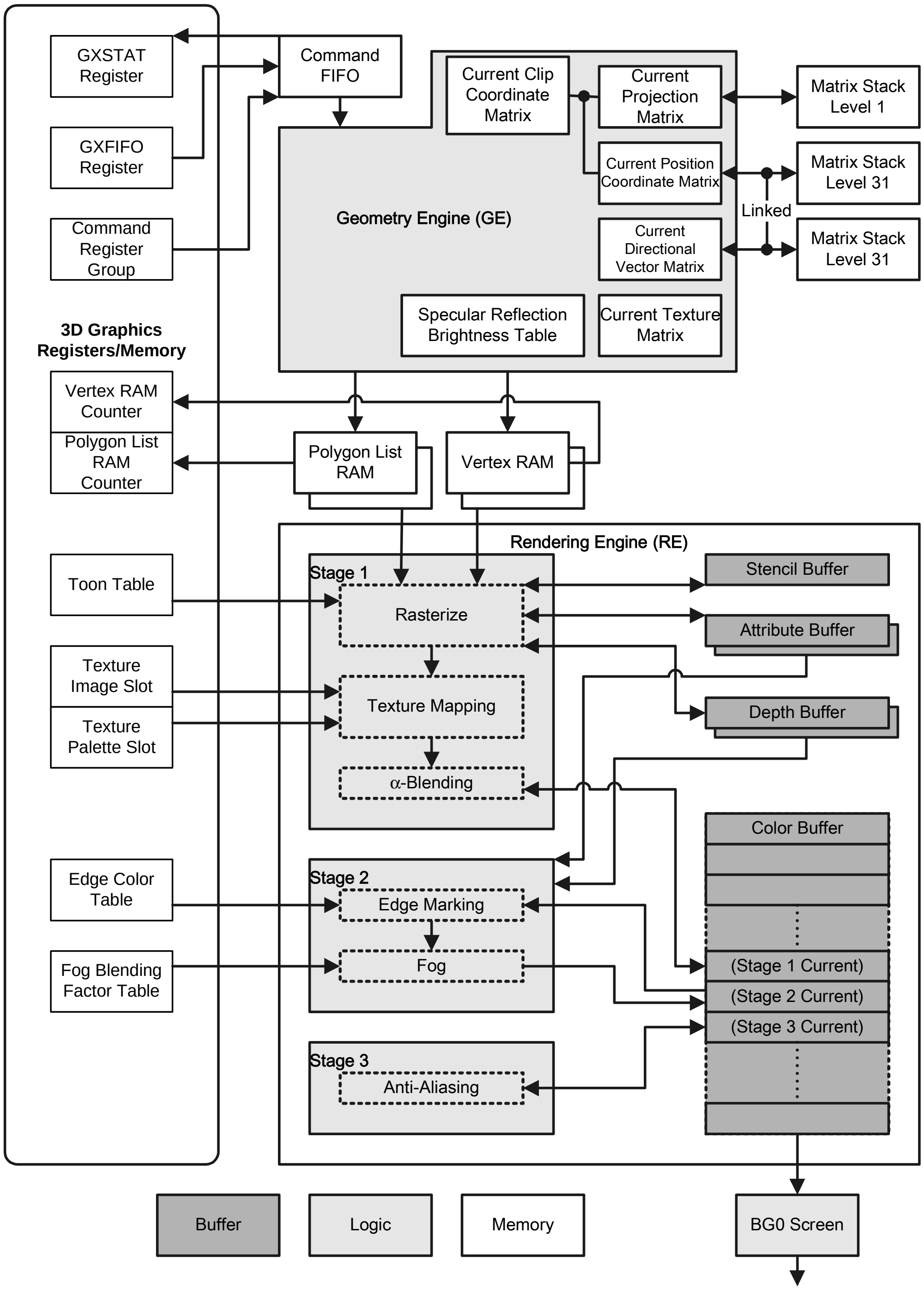


7 3D Graphics

The TWL 3D graphics engine is unchanged from NITRO except for the fact that known issues in NITRO can be corrected with system configuration. For details on the revisions, refer to "7.2 Geometry Engine" on page 182, "7.3 Rendering Engine" on page 251, and "2 Differences from NITRO" on page 9.

Figure 7-1 is a schematic of the hardware blocks involved in the rendering of 3D graphics.

Figure 7-1 : 3D Graphics Hardware Block Diagram



- Polygon List RAM, Vertex RAM

  The data that is passed from the Geometry Engine to the Rendering Engine is stored in Polygon List RAM and Vertex RAM.

  Table 7-1 shows the capacity for Polygon list RAM and Vertex RAM.

Table 7-1 : Capacity of Polygon List RAM and Vertex RAM

RAM	Capacity
Polygon List RAM	2048 polygons
Vertex RAM	6144 vertices

  The capacity of polygon list RAM is just enough for 2048 triangular polygons. Because quadrilateral polygons have four vertices, fewer of these can be stored in this RAM. A maximum of 1706 can be stored for the quadrilateral strip because neighboring quadrilateral polygons share vertices.

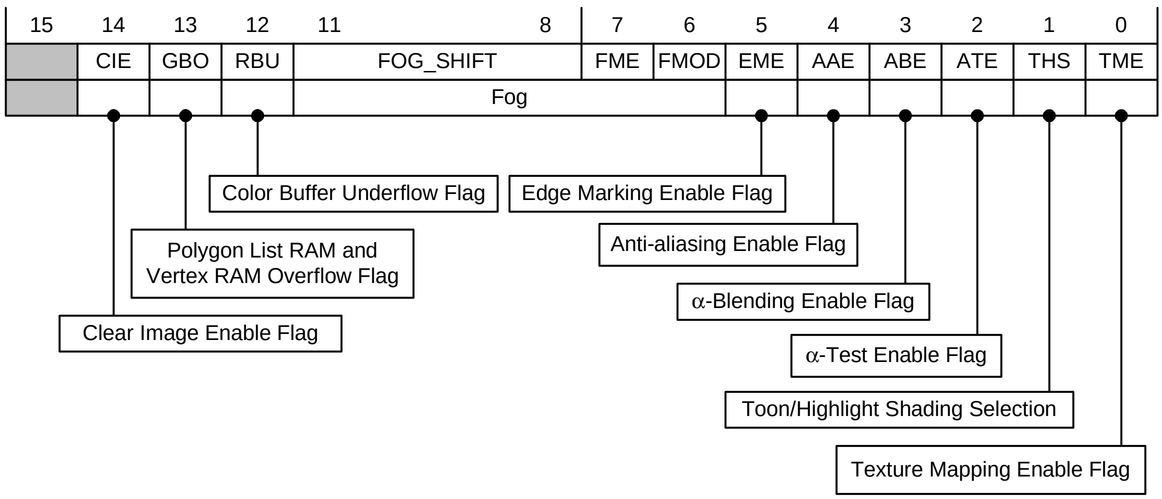
- Buffers inside the Rendering Engine

  The Stencil buffer, Attribute buffer, Depth buffer, and Color buffer are memory regions that store information for each pixel. One buffer block in Figure 7-1 depicts a line buffer for an LCD horizontal line of 256 pixels. For more details, see the "7.3 Rendering Engine" on page 251.

7.1 3D Display Control

DISP3DCNT: 3D Display Control Register

Name: DISP3DCNT Address: 0x04000060 Attribute: R/W Initial value: 0x0000



- CIE[d14] : Clear Image enable flag

  Two VRAMs of 128 kilobytes each are used to set the values for Clear Color, Clear α, Clear Depth, and Clear Fog.

  The Clear α value is specified with 1 bit, so you can select only transparent or opaque.

  VRAM2 banks must be assigned to the Clear Image buffer with the RAM Control register 0.

  Even when this feature is used, the register value is used for the Clear Polygon ID.

  For further information, see "7.3.3.2 Initializing with Clear Images" on page 257.

- GBO[d13] : Polygon List RAM and Vertex RAM overflow flag

  The flag is set to 1 when the Geometry Engine processes too many polygons and vertices and Polygon List RAM and Vertex RAM overflow.

  Once an overflow has occurred, the bit remains 1 even if it is no longer overflowing. You can reset the flag by writing 1.

0	No overflow
1	Overflow

  When the polygon list RAM or vertex RAM overflow, the polygon that caused the overflow and all the polygons after it are not registered.

- RBU[d12] : Color buffer underflow flag

  This flag is set to 1 when rendering is not done in time to display and a Color buffer underflow occurs (a *line overflow* in the Rendering Engine).

  Once the Color buffer has underflowed, this bit remains 1 even upon deletion. You can reset by writing 1. See "7.3.2 Rendering Methods" on page 253.

0	No underflow
1	Underflow

- [d11–d06] : Fog

  This feature applies a fog effect. For details, see the "7.3.1 Overview" on page 251.

  - [d11–d08] : Fog Shift

    The depth value used by fog uses the upper 15 bits (called the fog depth value) of the 24 bits.

    The Fog table is referenced using 5 bits of the depth value as an index.

    When the Fog Shift is 0, the bits d14 - d10 of the depth value are referenced as the index. For every 1-step increase in the Fog Shift, the reference bits are shifted 1-bit to the right.

    Note: Setting values of 11–15 is prohibited.

  - FME[d07] : Fog master enable flag

0	Disable
1	Enable

  - FMOD[d06] : Fog mode

    Selecting 1 makes 3D objects appear to dissolve into a 2D background.

0	Fog blending using pixel's color value and α value.
1	Fog blending using only the pixel's α value.

- EME[d05] : Edge-marking enable flag

  This feature draws an outline in the specified color around the edges of polygons with different polygon IDs.

  For details, see the "7.3.1 Overview" on page 251.

0	Disable
1	Enable

- AAE[d04] : Anti-aliasing enable flag

  This feature blends the edges of a polygon with the color value for the polygon behind it.

  For details, see the "7.3.1 Overview" on page 251.

  If you plan to capture the rendering result used as Bitmap OBJ, etc., disabling this flag gives you more natural images.

0	Disable
1	Enable

- ABE[d03] : Alpha-Blending enable flag

  This feature blends the Color buffer's color with the fragment color in accordance with the fragment's alpha value.

  For details, see the "7.3.1 Overview" on page 251.

0	Disable
1	Enable

- ATE[d02] : α-Test enable flag

  This feature enables you to skip the drawing of pixels that have an α-value lower than the specified value.

  For details, see the "7.3.1 Overview" on page 251.

0	Disable
1	Enable

- THS[d01] : Toon/Highlight Shading Selection

  This bit selects the shading mode for the polygon specified for Toon shading/Highlight shading with the PolygonAttr command.

  - TMOD : Toon/Highlight polygon mode

0	Toon shading
1	Highlight shading

- TME[d00] : Texture Mapping master enable flag

  This selects whether to perform texture mapping.

0	Disable
1	Enable

7.2 Geometry Engine

In TWL mode, it is possible to use the revised circuits for the following features that had problems on NITRO by means of the system configuration. However, in NITRO compatibility mode, it will be necessary to follow the notes to work around the problems.

- Command FIFO : Refer to "7.2.6 Geometry Commands" on page 191.
- 1-Dot Polygons : Refer to "7.2.12 Polygon Attributes" on page 221.
- Clipping Feature : Refer to "7.2.13 Polygons" on page 225.
- Texture Image Parameters : Refer to "7.2.14 Texture Mapping" on page 231.
- BoxTest : Refer to "7.2.15 Tests" on page 241.

7.2.1 Overview

Table 7-2 lists geometry engine specifications.

Table 7-2 : Geometry Engine Specifications

Operating Frequency	33.514MHz
Coordinate Transformation	Maximum 4 million vertices/sec
Matrix Computation	4×4 matrix computation and matrix stack
Clipping	6-plane clipping
Lighting	Light: Parallel light source × 4 Material: Reflected light (diffuse reflection, specular reflection, ambient reflection), Emission light
Other features	Backface culling; Specify display of 1-dot polygons (see note); Box culling test; Texture coordinate transformation; Adjust specular reflection shininess distribution

Note: A *1-dot polygon* is a polygon whose constituent coordinates (x, y) have been condensed to the same coordinate.

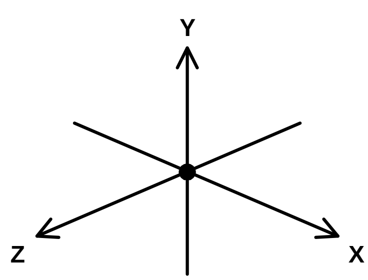
7.2.2 Coordinate System

In three-dimensional space, the coordinate system can be defined in two ways: as a right-handed coordinate system or as a left-handed coordinate system, depending on the direction of the z-axis relative to the x and y axes.

As a rule, TWL adopts the right-handed coordinate system. However, because Z values are inverted with the projection matrix, the coordinates after clipping are in the left-handed coordinate system.

Figure 7-2 shows the relation of the x, y, and z axes in a right-handed coordinate system.

Figure 7-2 : Right-Handed Coordinate System



7.2.3 Coordinate Transformations

In OpenGL, when a model is located in view coordinates, the model is first multiplied by the model view matrix, which includes the view transformation. This is then multiplied by the projection matrix, and the apparent size of the model is determined based on the view volume.

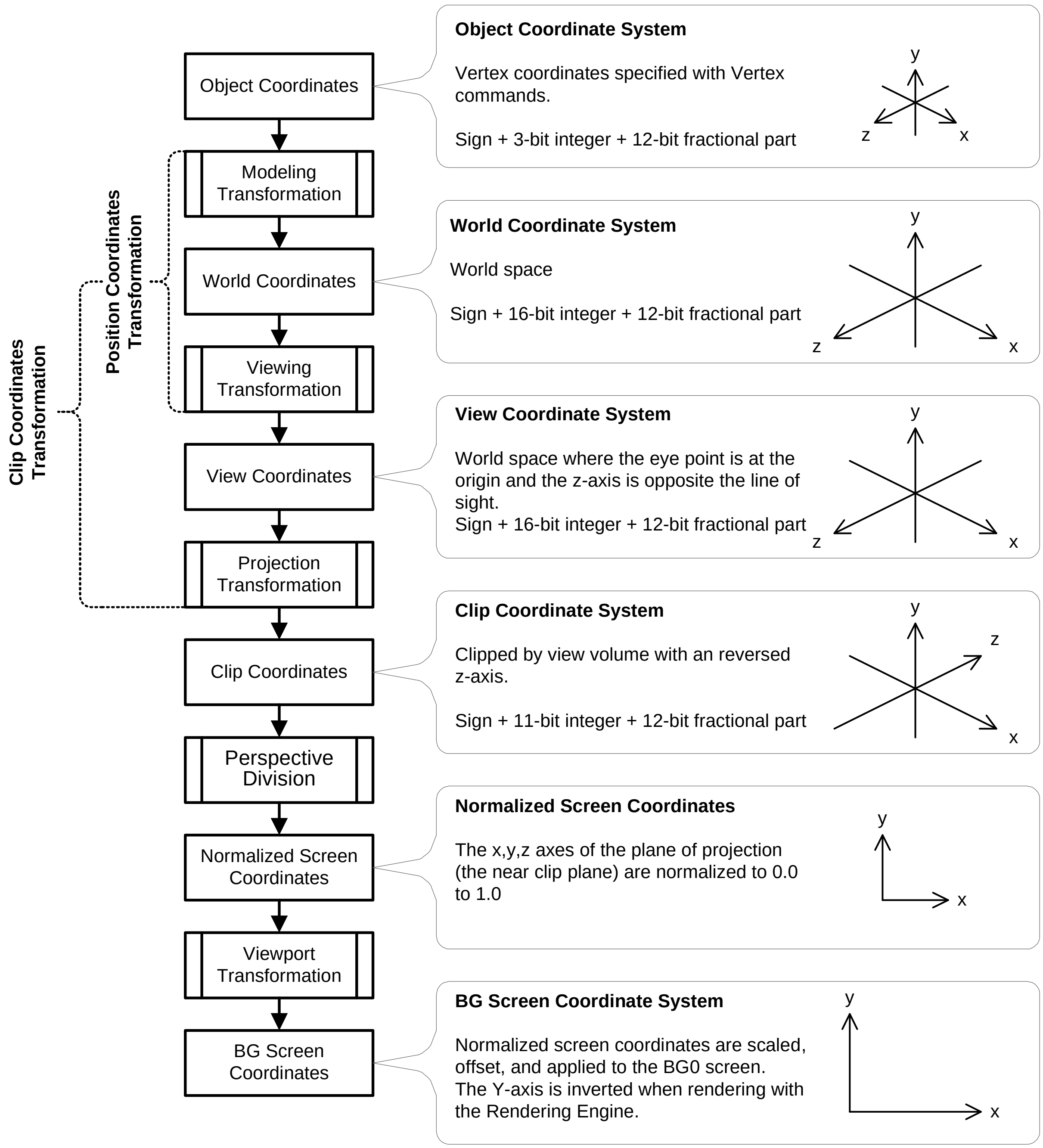
To reduce the load on the hardware, TWL uses a clip coordinate matrix that is a concatenation of the projection matrix and the position coordinates matrix so that clip coordinate conversion is done with only one coordinate transformation. Then the clip coordinate values (x, y, z, w) are divided by 2w (perspective division) after only the w coordinate is translated to get the normalized screen coordinates, and a scaling transformation is done on the BG screen coordinates by a viewport transformation.

For the vertex's normal vector and light vector, OpenGL performs a transformation with the transposed matrix of the modelview matrix. In contrast, TWL assumes the vector is normalized and uses only the rotational component's matrix (the orthogonal matrix) for the transformation. This kind of transformation is called a *directional vector transformation*.

In OpenGL, vertex position coordinates and directional vectors are transformed into view coordinates just by setting the modelview matrix. But in TWL, the vertex position coordinates and directional vectors are transformed by separate matrices, and these are separately defined as the *position coordinates matrix* and *directional vector matrix*.

Figure 7-3 illustrates the coordinate transformation flow on the TWL.

Figure 7-3 : Coordinate Transformation Flow Chart



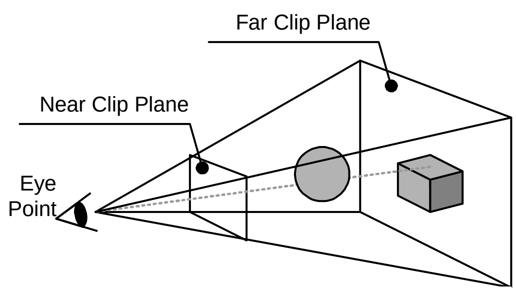
7.2.4 Projection Transformations

The perspective of the viewed polygon object from the eye point is defined by the view volume.

The view volumes and projection matrices for perspective projections (Figure 7-4) and orthogonal projections ("Figure 7-5 : Orthogonal Projections" on page 186) are described below.

1. Perspective Projections

Figure 7-4 : Perspective Projections



a. Left-Right Asymmetrical Perspective Projection

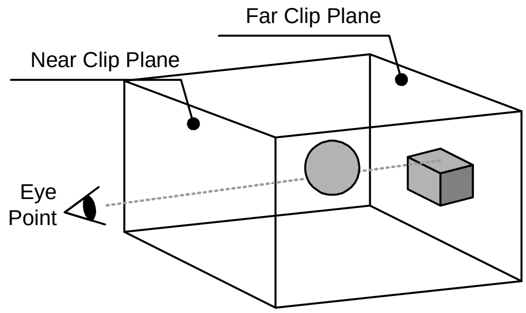
$$Frustum = \begin{bmatrix} \frac{2n}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2n}{t-b} & 0 & 0 \\ \frac{r+l}{r-l} & \frac{t+b}{t-b} & -\frac{f+n}{f-n} & -1 \\ 0 & 0 & -\frac{2fn}{f-n} & 0 \end{bmatrix} \times scaleW$$

b. Left-Right Symmetrical Perspective Projection

$$Perspective = \begin{bmatrix} \frac{\cos\theta}{asp \cdot \sin\theta} & 0 & 0 & 0 \\ 0 & \frac{\cos\theta}{\sin\theta} & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -1 \\ 0 & 0 & -\frac{2fn}{f-n} & 0 \end{bmatrix} \times scaleW$$

2. Orthogonal Projections

Figure 7-5 : Orthogonal Projections



$$Ortho = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & -\frac{2}{f-n} & 0 \\ -\frac{r+l}{r-l} & -\frac{t+b}{t-b} & -\frac{f+n}{f-n} & 1 \end{bmatrix}$$

t : Top edge y-coordinate of near clip plane

b : Bottom edge y-coordinate of near clip plane

r : Right edge x-coordinate of near clip plane

l : Left edge x-coordinate of near clip plane

n : Distance from eye point to near clip plane

f : Distance from eye point to far clip plane

θ : Angle of field-of-view (screen angle) in vertical (y) direction ÷ 2

asp : Aspect ratio of width to height of field-of-view (height:width ratio = width of field-of-view ÷ height of field-of-view)

scaleW : Parameter for precision-adjusting view volume

(Use to change the scaling of clip coordinate space and increase the precision of the orthogonal screen coordinates after perspective division.)

7.2.5 Depth Buffering

1. For Perspective Projections

The perspective projection matrix parameters are set as shown below. (For details about elements p0-p5, see "7.2.4 Projection Transformations" on page 185).

$$M = \begin{bmatrix} p0 & 0 & 0 & 0 \\ 0 & p2 & 0 & 0 \\ p1 & p3 & p4 & -1 \\ 0 & 0 & p5 & 0 \end{bmatrix} \times scaleW$$

When the View coordinates are (x, y, z, 1), the Clip coordinates (x', y', z', w') are as follows:

$$x' = (p0 \times x + p1 \times z) \times scaleW$$
$$y' = (p2 \times y + p3 \times z) \times scaleW$$
$$z' = (p4 \times z + p5 \times 1) \times scaleW$$
$$w' = -z \times scaleW$$

When each clip coordinate component is translated by w' and then divided by 2w' (perspective division), the normalized screen coordinates (x", y", z", 1) are obtained. The z" component is calculated as follows:

$$z'' = \frac{z' + w'}{2w'} = \frac{(p4 \times z + p5 - z) \times scaleW}{-2z \times scaleW} = \frac{1}{2} - \frac{p4}{2} - \frac{p5}{2z}$$

p4 and p5 are elements of the perspective projection matrix, so using the matrix shown in the equation "a. Left-Right Asymmetrical Perspective Projection" on page 185 yields:

$$z'' = \frac{f}{(f-n)}\left(1 + \frac{n}{z}\right) \qquad (-far \le z \le -near)$$

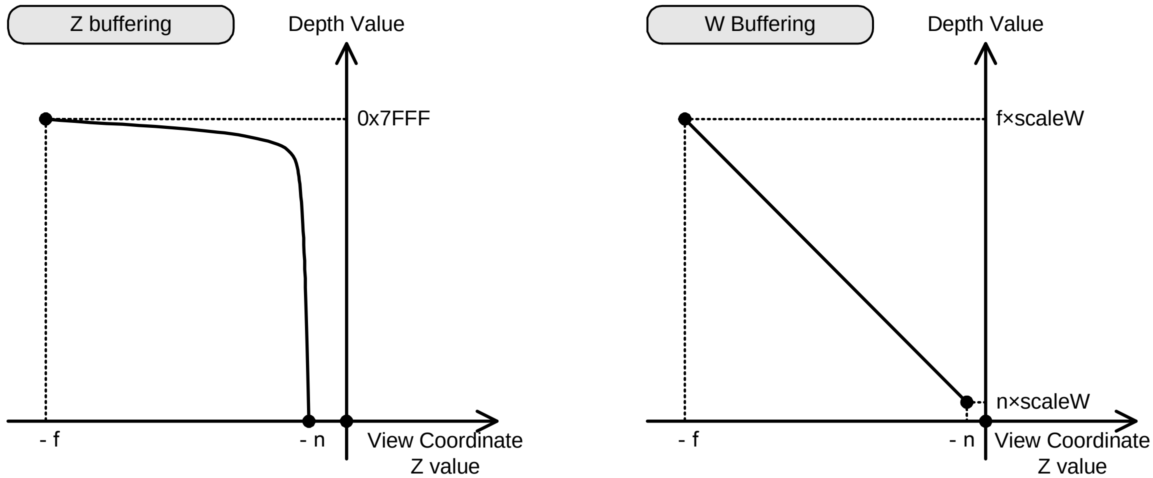
During Z buffering, this z" value is multiplied by 0x7FFF to get the depth value z'". The z'" value is proportional to the inverse of the View coordinate z value, so in View coordinate space the position coordinates are more precise the closer they are to the eye point and less precise the farther they are away from the eye point. As a result, when you represent a large space, drawings that are farther away tend to be more imprecise. To resolve this problem, TWL supports *W buffering*, which uses clip coordination as the depth value.

Because the depth value is taken as a multiple of the view coordinate z from the equation...

$$w' = -z \times scaleW \qquad (-far \le z \le -near)$$

... the precision of rendering in the distance improves. The trade-off is that this diminishes the precision of nearby images compared to Z-buffering. But by enlarging the view volume space and adjusting scaleW in order to maintain a certain level of resolution, this should not present a problem for nearby images. (See Figure 7-6.)

Figure 7-6 : Z-Buffering and W-Buffering (Perspective Projection)



f : Distance from eye point to far clip plane

n : Distance from eye point to near clip plane

2. For Orthogonal Projections

The orthogonal projection matrix parameters are set as shown below. (For details about elements p0-p5, see "7.2.4 Projection Transformations" on page 185.)

$$M = \begin{bmatrix} p0 & 0 & 0 & 0 \\ 0 & p2 & 0 & 0 \\ 0 & 0 & p4 & 0 \\ p1 & p3 & p5 & 1 \end{bmatrix} \times scaleW$$

When the View coordinates are (x, y, z, 1), the Clip coordinates (x', y', z', w') are as follows:

$$x' = (p0 \times x + p1 \times 1) \times scaleW$$
$$y' = (p2 \times y + p3 \times 1) \times scaleW$$
$$z' = (p4 \times z + p5 \times 1) \times scaleW$$
$$w' = 1 \times scaleW$$

When each component of the clip coordinate is translated by w' and then divided by 2w'

When each clip coordinate component is translated by w' and then divided by 2w' (perspective division), the normalized screen coordinates (x", y", z", 1) are obtained. When scaleW = 1, coordinates are translated by 1 and divided by 2, so the clip coordinate system values -1.0 to 1.0 are transformed into the normalized screen coordinate system values 0.0 to 1.0.

The normalized screen coordinate z" component is calculated as follows:

$$z'' = \frac{z' + w'}{2w'} = \frac{(p4 \times z + p5 + 1) \times scaleW}{2 \times scaleW} = \frac{p4 \times z + p5 + 1}{2}$$

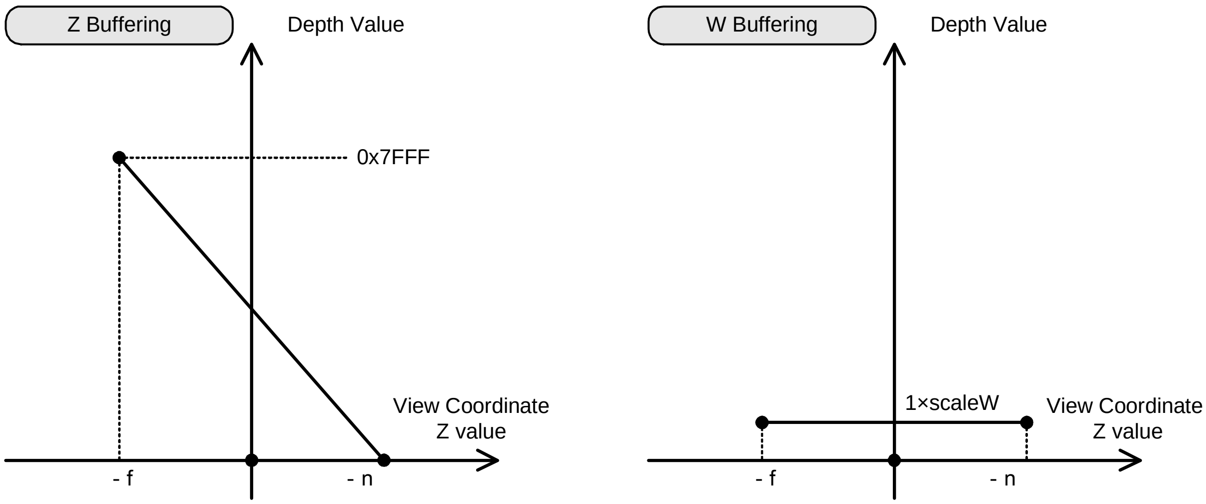
p4 and p5 are elements of the orthogonal projection matrix, so using the matrix shown in the equation "2. Orthogonal Projections" on page 186 yields:

$$z'' = \frac{-1}{(f - n)}(z + n) \qquad (-far \le z \le -near)$$

During Z buffering, this z" value is multiplied by 0x7FFF to get the depth value z'''. This z''' value is proportional to the View coordinate z value, so no problems arise with the precision of distant images.

When w buffering is used, the w' clip coordinate which serves as the depth value is always fixed to the value 1 x scaleW. For this reason, w buffering is not used with orthogonal projections. (See Figure 7-7.)

Figure 7-7 : Z-Buffering and W-Buffering (Orthogonal Projection)



f : Distance from eye point to far clip plane

n : Distance from eye point to near clip plane

Depth value format

The TWL Depth buffer uses 24 bits for each pixel, so the depth value must fit inside that range.

For Z buffering, the depth value is the value that expresses the distance from the near clip plane to the far clip plane with 24-bit precision.

For W buffering, the distance from the eye point in the View coordinate system must fit within the 24-bit precision (sign + 11-bit integer + 12-bit fractional part) range of the Clip coordinate system, and the depth value is the result of translating the W value of the clip coordinates a distance of W and then dividing by two (12-bit integer + 12-bit fractional part).

Note: Make sure that the Clip coordinate values do not exceed the 24-bit range.

For fog, the depth value is the upper 15 of the 24 bits. See "7.3.3.1 Initializing with the Clear Registers" on page 256 for details.

7.2.6 Geometry Commands

To transfer the data of matrices and polygons, etc. to the Geometry Engine requires writing command strings to Command FIFO. There are two ways to write to Command FIFO:

Method 1: Write parameters to the group of command registers mapped in the register space of the main processor.

Write the parameters into Command registers, and the command code and parameters are automatically written into Command FIFO. This method works when the CPU will process one command at a time.

Method 2: Write command code and parameters to the Command FIFO register.

This method is appropriate for transferring large amounts of data to the Geometry Engine, such as for DMA transfers of command strings stored in main memory.

Note: The Geometry Command register group and Command FIFO are specialized for 32-bit access. Whether you are writing using the CPU or DMA, make sure the access width is 32 bits.

When an attempt is made to write to Command FIFO when it is full, the process enters a wait state until 32 bits open up in Command FIFO. During this wait state the bus cannot be used even by another bus master. You can avoid this situation by doing the following:

Transfer commands using the DMA Geometry Command FIFO startup mode

In this mode, DMA is started when Command FIFO becomes less than half full, sending in units of 112 words (see note) until the specified transfer volume is reached. See "8 DMA" on page 297.

Note: If commands have been packed, the word count equals the number of words before unpacking.

You can determine the status of Command FIFO by checking the Command FIFO status flag in the Geometry Engine status register (GXSTAT).

GXFIFO: Command FIFO Register

Name: GXFIFO   Address: 0x04000400   (Image: 0x04000404 - 0x0400043F)   Attribute: W

31 – 24	23 – 16	15 – 8	7 – 0

Data written to the Command FIFO register is sent to Command FIFO.

Command FIFO has a depth of 32 bits x 256 levels.

Be careful not to transfer undefined command codes.

Command Packing

When transferring command strings to the Command FIFO register, the command strings can be compressed by packing up to four command codes in one word.

Packed command strings are first decompressed and then stored in command FIFO. The packed command strings are stored in command FIFO in order starting from the low-order byte, so you need to pack the command codes starting from the low-order address and fill the empty higher-order bytes with 0.

Figure 7-8 shows the different command transfers for commands that are packed and not packed.

Figure 7-8 : Transferring Packed and Non-Packed Commands

Non-packed Commands

Word Count	GXFIFO			
	d31 - d24	d23 - d16	d15 - d08	d07 - d00
1	0x00	0x00	0x00	Command 1
2	Parameter for Command 1			
3	0x00	0x00	0x00	Command 2
4	Parameter A for Command 2			
5	Parameter B for Command 2			
6	0x00	0x00	0x00	Command 3
7	0x00	0x00	0x00	Command 4
8	Parameter for Command 4			
9	0x00	0x00	0x00	Command 5
10	Parameter for Command 5			
11	0x00	0x00	0x00	Command 6

Packed Commands

GXFIFO			
d31 - d24	d23 - d16	d15 - d08	d07 - d00
Command 4	Command 3	Command 2	Command 1
Parameter for Command 1			
Parameter A for Command 2			
Parameter B for Command 2			
Parameter for Command 4			
0x00	0x00	Command 6	Command 5
Parameter for Command 5			
4-Word Reduction			

In the example shown above, Commands 1, 4, and 5 have one parameter each, Command 2 has two parameters, and Commands 3 and 6 have no parameters.

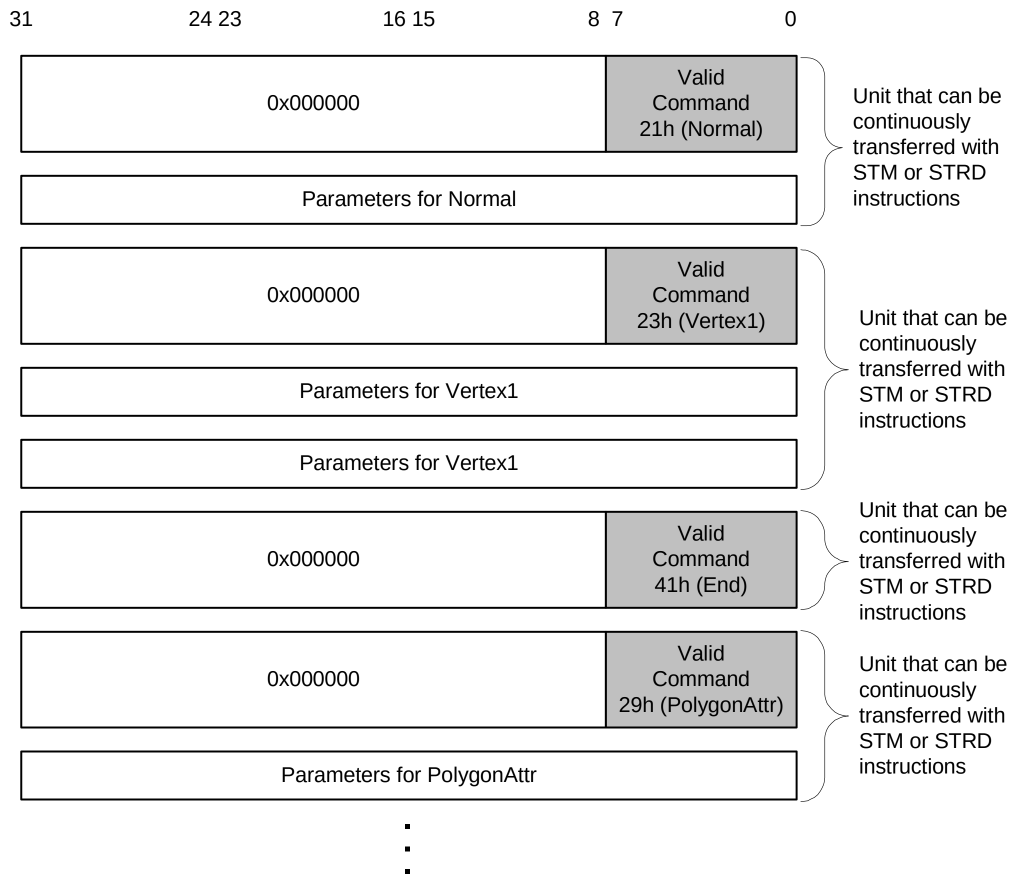
In Figure 7-8, a data volume of 11 words is sent to the Command FIFO register when the commands are not packed, but when commands are packed only 7 words are sent, for a savings of 4 words.

Precautions regarding the CPU continuously writing to the Geometry FIFO

Continuous writing to the Geometry FIFO using STM or STRD instructions can occur properly only when the two following conditions are met. See Figure 7-9.

- Command Pack is not used
- Write only one command-parameter pair at a time

Figure 7-9 : Continuous Writing to the Geometry FIFO Using STM or STRD Instructions



The "Unit that can be continuously transferred with STM or STRD instructions" mentioned above can be written at a single time. However, do not perform a write with STM or STRD instructions that exceed this unit. Leave a blank interval of one system cycle between each "Unit that can be continuously transferred with STM or STRD instructions."

Cautions regarding Data Arrays for Command Packs

Writing to the Geometry FIFO when command packs are used can occur properly only when one of the following conditions are met whether using the CPU or DMA.

- Do not have a command without parameters (see Note 2 below) come in the top level of valid commands in the command pack (see Note 1 below). See Figure 7-10 and Figure 7-11 for details.

Note 1: *Valid command* indicates a command defined within the region between 0x10 and 0xff. *Invalid commands* are in the region between 0x00 and 0x0f.

Note 2: A "command without parameters" is one of the four following commands:

- PushMatrix
- LoadIdentity
- End
- Commands undefined within the region between 0x10 and 0xff

Figure 7-10 : Case 1: Preventing a Command Without Parameters from Being the First Valid Command

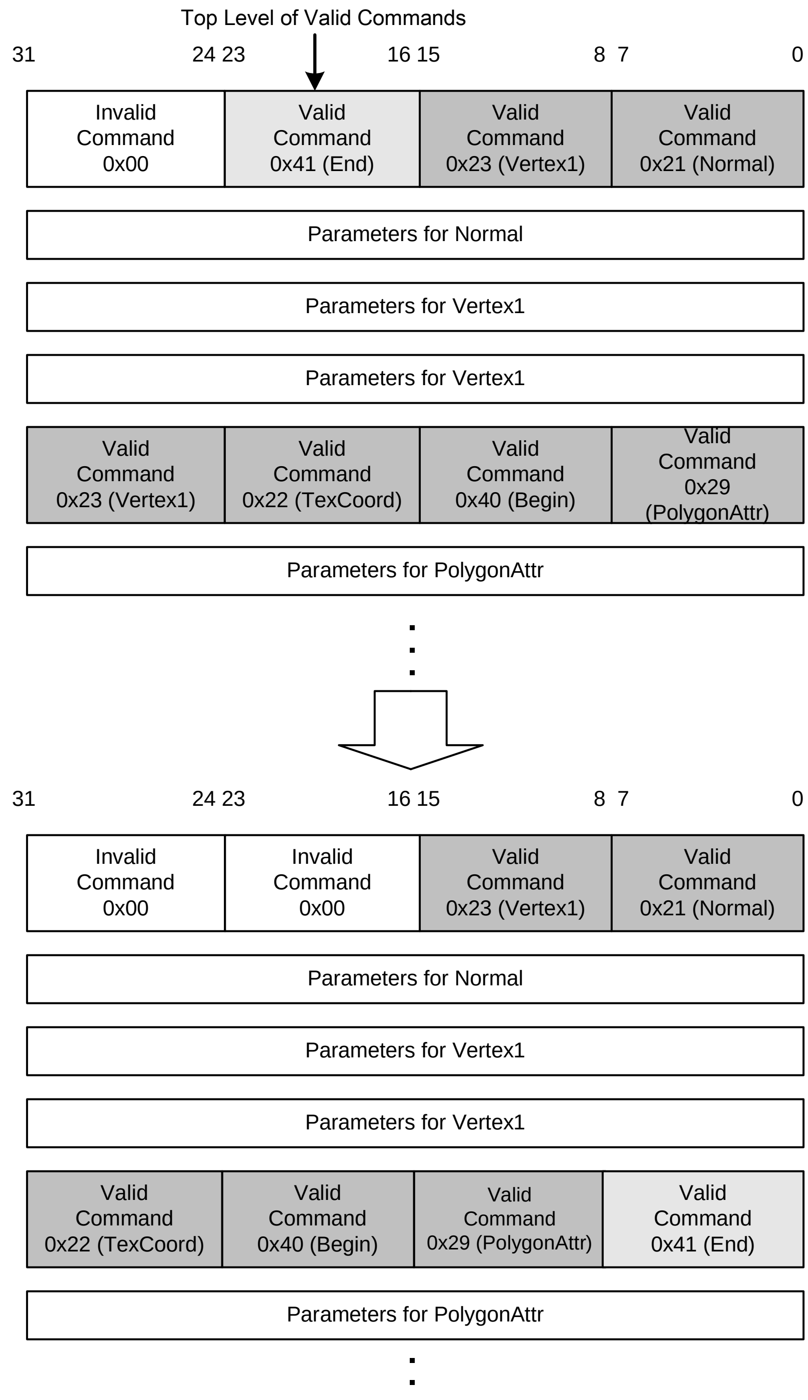
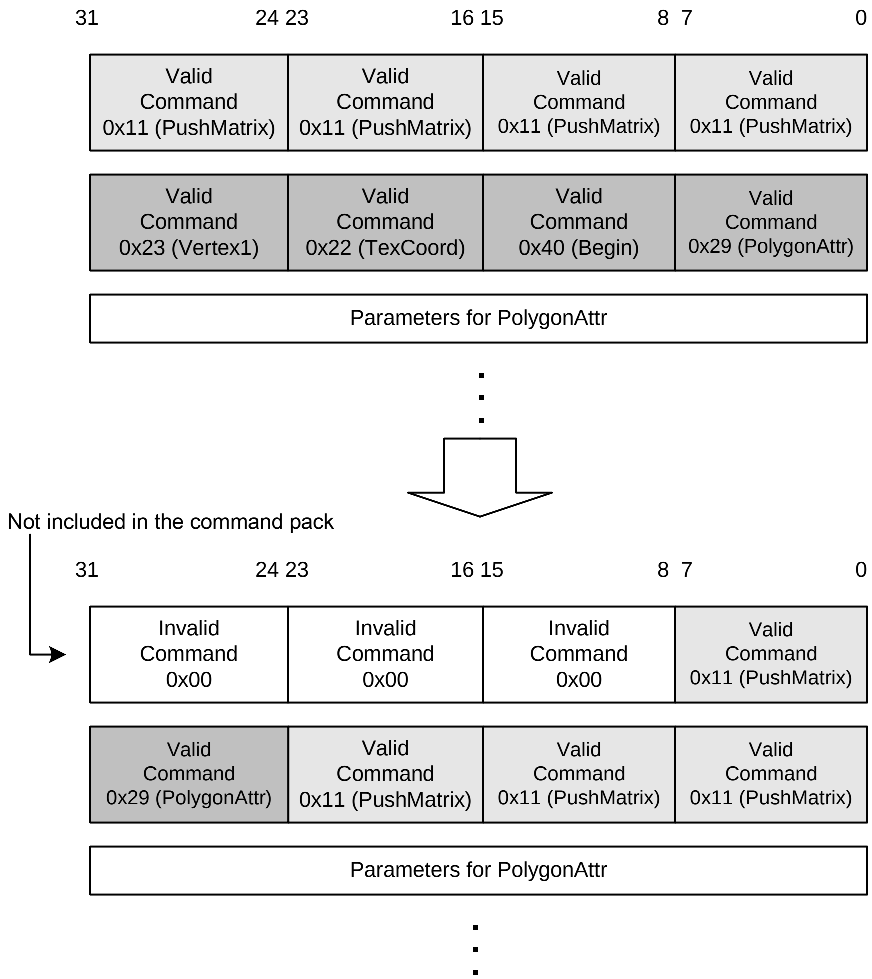


Figure 7-11 : Case 2: Preventing a Command Without Parameters from Being the First Valid Command



- When a command without parameters comes in the top level of valid commands in the command pack, insert zeros at the end of the parameter array corresponding to that command pack.

Figure 7-12 : When the First Valid Command Has No Parameters

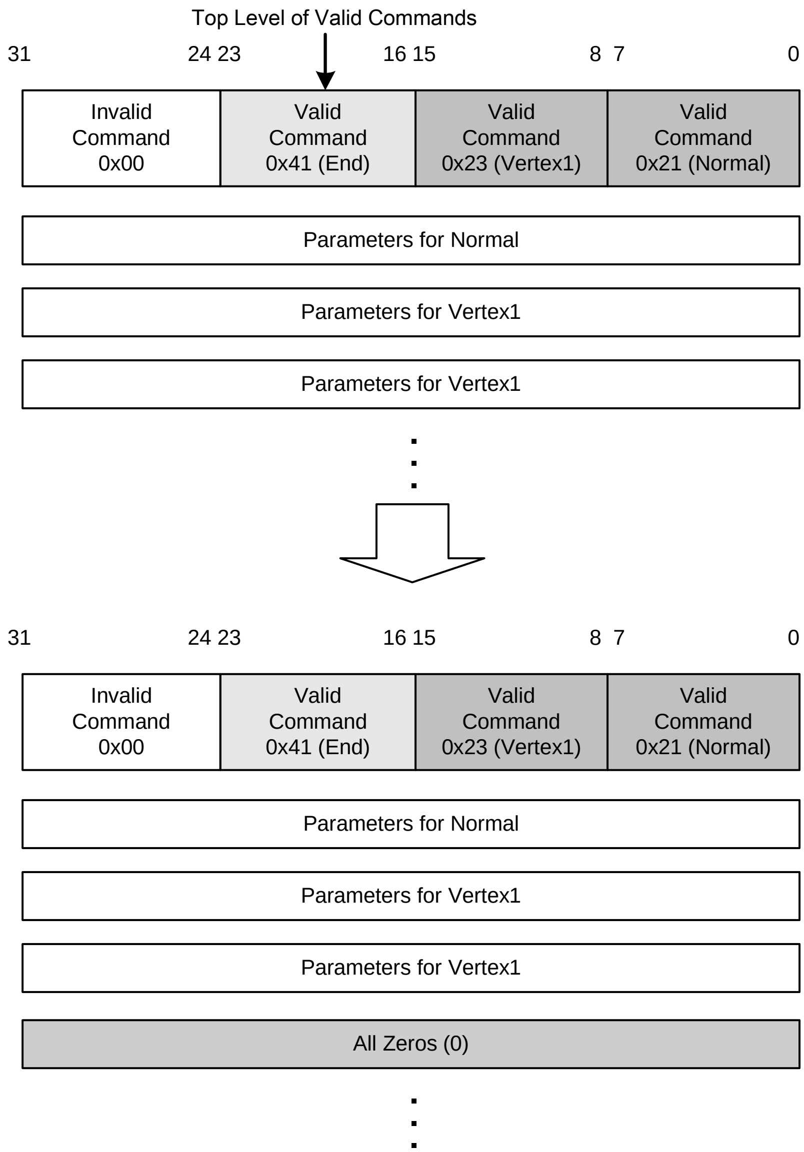


Table 7-3 and Table 7-4 show lists of Geometry Commands.

Table 7-3 : Geometry Commands (in Command Code Order)

Category	Feature	Command name	Command register address (see note)	Command code	No. of words in parameter	Page
—	No operation	Nop	—	0x00	0	None
Matrix mode	Sets the matrix mode	MatrixMode	0x440	0x10	1	206
Operations on the current matrix	Pushes to stack	PushMatrix	0x444	0x11	0	211
	Pops from stack	PopMatrix	0x448	0x12	1	211
	Writes to specified location in stack	StoreMatrix	0x44C	0x13	1	212
	Reads from specified location in stack	RestoreMatrix	0x450	0x14	1	212
	Initializes a unit matrix	Identity	0x454	0x15	0	207
	Sets a 4x4 matrix	LoadMatrix44	0x458	0x16	16	207
	Sets a 4x3 matrix	LoadMatrix43	0x45C	0x17	12	207
	Multiplies a 4x4 matrix	MultMatrix44	0x460	0x18	16	208
	Multiplies a 4x3 matrix	MultMatrix43	0x464	0x19	12	208
	Multiplies a 3x3 matrix	MultMatrix33	0x468	0x1A	9	209
	Multiplies a scale matrix	Scale	0x46C	0x1B	3	210
	Multiplies a translation matrix	Translate	0x470	0x1C	3	209
Vertex information	Directly sets vertex color	Color	0x480	0x20	1	226
	Sets normal vector	Normal	0x484	0x21	1	227
	Sets texture coordinates	TexCoord	0x488	0x22	1	231
Vertex coordinates	Sets the vertex coordinates	Vertex	0x48C	0x23	2	227
	Same as above	Vertex10	0x490	0x24	1	228
	Sets the XY coordinates of the vertex	VertexXY	0x494	0x25	1	228
	Sets the XZ coordinates of the vertex	VertexXZ	0x498	0x26	1	228
	Sets the YZ coordinates of the vertex	VertexYZ	0x49C	0x27	1	228
	Sets vertex using the differential value of the last-set coordinate	VertexDiff	0x4A0	0x28	1	229
Polygon attribute	Sets the polygon attribute	PolygonAttr	0x4A4	0x29	1	221
Texture information	Sets the texture parameters	TexImageParam	0x4A8	0x2A	1	232
	Sets the base address of the texture palette	TexPlttBase	0x4AC	0x2B	1	237
Material	Sets the colors for ambient reflection and diffuse reflection	MaterialColor0	0x4C0	0x30	1	217
	Sets the colors for emission light and specular reflection	MaterialColor1	0x4C4	0x31	1	217
	Sets the specular reflection shininess table	Shininess	0x4D0	0x34	32	218

Category	Feature	Command name	Command register address (see note)	Command code	No. of words in parameter	Page
Light	Sets the directional vector for light	LightVector	0x4C8	0x32	1	214
	Sets the light color	LightColor	0x4CC	0x33	1	214
Vertex list begin/end	Declares the start of the vertex list	Begin	0x500	0x40	1	225
	Declares the end of the vertex list	End	0x504	0x41	0	226
Swap Rendering Engine reference data	Swaps the data group referenced by the Rendering Engine	SwapBuffers	0x540	0x50	1	203
Viewport	Sets the viewport	ViewPort	0x580	0x60	1	205
Test	Tests whether the box is inside the view volume	BoxTest	0x5C0	0x70	3	241
	Sets position coordinates for test	PositionTest	0x5C4	0x71	2	243
	Sets directional vector for test	VectorTest	0x5C8	0x72	1	243

Note: The Command register address values shown here are offset from address 0x04000000.

Be careful not to issue undefined command codes to the Geometry Engine's command FIFO.

Table 7-4 : Number of Geometry Command Run Cycles & Timing Related to Command Issue (in Command Code Order)

Command Name	Run cycle number	Issue timing	When settings take effect	When settings are destroyed
Nop		No Restriction		
MatrixMode	1	No Restriction	When command is executed	When next MatrixMode command is executed
PushMatrix	17			When next Push/StoreMatrix command is executed
PopMatrix	36			When next Matrix change command is executed
StoreMatrix	17			When next Push/StoreMatrix command is executed
RestoreMatrix	36			When next Matrix change command is executed
Identity	19			
LoadMatrix44	34			
LoadMatrix43	30			
MultMatrix44	35*1			
MultMatrix43	31*1			
MultMatrix33	28*1			
Scale	22			
Translate	22*1			
Color	1			When next Color, Normal command is executed
Normal	9 – 12 (+2)*2, 5			
TexCoord	1(+1)*5			When next TexCoord command is executed
Vertex	9(+2)*5	Only between Begin–End		When next Vertex related command is executed
VertexShort	8(+2)*5			
VertexXY	8(+2)*5			
VertexXZ	8(+2)*5			
VertexYZ	8(+2)*5			
VertexDiff	8(+2)*5			
PolygonAttr	1	Only outside Begin – End	When Begin command is executed (settings are valid between Begin–End units)*3	When next PolygonAttr command is executed
TexImageParam	1	Per Polygon*4	When command is executed (settings are valid in polygon units)	When next TexImageParam command is executed
TexPlttBase	1			When next TexPlttBase command is executed

<table>
<tr><th>Command Name</th><th>Run cycle number</th><th>Issue timing</th><th>When settings take effect</th><th>When settings are destroyed</th></tr>
<tr><td>MaterialColor0</td><td>4</td><td rowspan="7">No Restriction</td><td rowspan="5">When Normal command is executed</td><td>When next MaterialColor0 command is executed</td></tr>
<tr><td>MaterialColor1</td><td>4</td><td>When next MaterialColor1 command is executed</td></tr>
<tr><td>Shininess</td><td>32</td><td>When next Shininess command is executed</td></tr>
<tr><td>LightVector</td><td>6</td><td>When next LightVector command is executed</td></tr>
<tr><td>LightColor</td><td>1</td><td>When next LightColor command is executed</td></tr>
<tr><td>Begin</td><td>1</td><td rowspan="2">When command is executed</td><td>When next Begin command is executed</td></tr>
<tr><td>End</td><td>1</td><td>No set value</td></tr>
<tr><td>SwapBuffers</td><td>392</td><td rowspan="3">Only outside Begin – End</td><td>When enter V-Blank period</td><td>When next SwapBuffers command is executed</td></tr>
<tr><td>ViewPort</td><td>1</td><td rowspan="4">When command is executed</td><td>When next ViewPort command is executed</td></tr>
<tr><td>BoxTest</td><td>103</td><td>When next BoxTest command is executed</td></tr>
<tr><td>PositionTest</td><td>9</td><td>Per Polygon*4</td><td>When next PositionTest is executed</td></tr>
<tr><td>VectorTest</td><td>5</td><td>No Restriction</td><td>When next VectorTest is executed</td></tr>
</table>

The number of run cycles is a system clock (33.514MHz) converted value, and is the minimum value. Extra cycles will be required if access collisions, clipping, pipeline hazards, or other complications occur.

*1 When in Position and Vector Simultaneous Setting mode, this takes another 30 cycles.

*2 Is increased according to the number of lights that are enabled (ON)

*3 The PolygonAttr command is enabled with the Begin command. However, to reflect the light enable flag on vertex color, issue a Normal command again to recalculate the lighting.

*4 Concerning the Polygon unit:

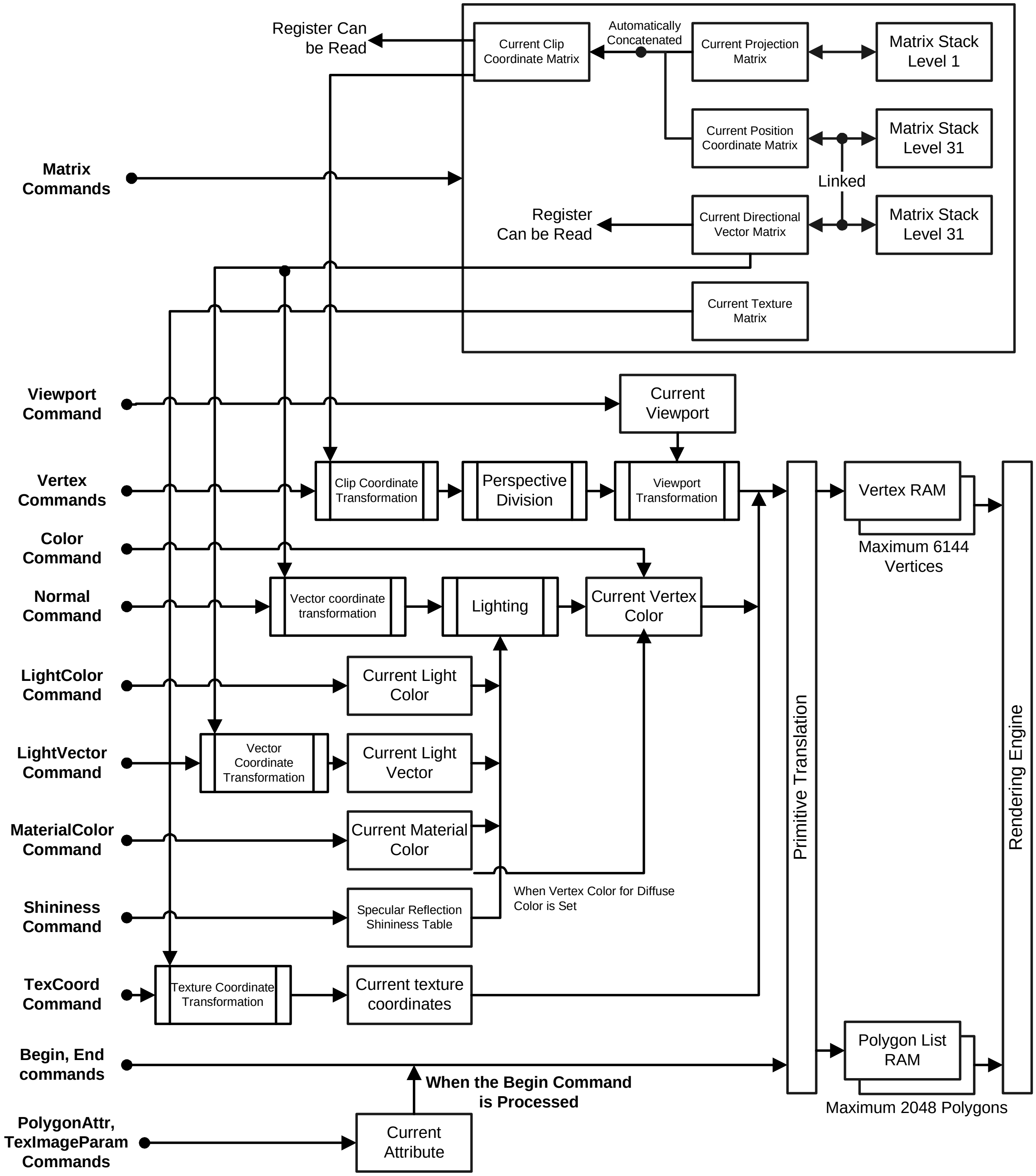
Commands that can be issued in polygon units can be issued in the Vertex-related command string at polygon breakpoints (breakpoints appear as ● in the following table.)

Triangular Polygons	Triangular Polygon Strip	Quadrilateral Polygons	Quadrilateral Polygon Strip
● Begin ● Vertex //Polygon 1 Vertex Vertex ● Vertex //Polygon 2 Vertex Vertex ● Vertex //Polygon 3 Vertex Vertex ● End ●	● Begin ● Vertex //Polygon 1 Vertex Vertex Vertex Vertex //Polygon 2 Vertex //Polygon 3 ● End ●	● Begin ● Vertex //Polygon 1 Vertex Vertex Vertex ● Vertex //Polygon 2 Vertex Vertex Vertex ● Vertex //Polygon 3 Vertex Vertex Vertex ● End ●	● Begin ● Vertex //Polygon 1 Vertex Vertex Vertex Vertex //Polygon 2 Vertex Vertex //Polygon 3 Vertex ● End ●

*[5] The number of run cycles for commands corresponding to each source increases by the amount shown in parentheses when performing texture coordinate conversion in texture coordinate conversion mode.

Figure 7-13 shows a schematic of the main Geometry Command processes.

Figure 7-13 : Schematic of the Main Geometry Command Processes



Note: The flow shown here for the TexCoord command is for when the texture coordinate transformation mode is set to TexCoord source.

7.2.7 Swapping the Rendering Engine's Reference Data

SwapBuffers: Swaps the Data Group referenced by the Rendering Engine

Name: SWAP_BUFFERS   Address: 0x04000540   Attribute: W   Command Code: 0x50

31 … 24	23 … 16	15 … 8	7 … 2	1	0
				DP	YS

- DP[d01] : Depth buffering selection flag

  Selects the value used for the depth test. To read how the depth value differs depending on the depth-buffering method, see "7.2.5 Depth Buffering" on page 187.

0	Select buffering with the Z value
1	Select buffering with the W value (Does not function properly for orthogonal projections)

- YS[d00] : Translucent polygon Y-sorting selection flag

  Translucent polygons are polygons with ($1 \le \alpha \le 30$) or mapped with a translucent texture.

  Select manual sort mode to specify the order of rendering, such as when using shadow volume.

0	Auto-sort mode
1	Manual sort mode

The Geometry Engine writes the data passed to the Rendering Engine to Polygon List RAM.

For translucent polygons, you can choose whether to sort the data and then write, or to write the data in the order the polygons are processed without sorting. In Auto-sort mode, polygons are sorted from the polygon with smallest maximum Y value on the LCD (see note) to the polygon with the largest maximum Y value. Polygons that share the same maximum Y value are sorted in order from the polygon with smallest minimum Y value.

In Manual sort mode, polygons are sorted according to the order in which they are sent to the Geometry Engine.

In Auto sort mode, the Rendering Engine does not reference polygons with minimum Y values larger than the scan line (a line that is being rendered) or polygons with maximum Y values smaller than the scan line. However, all polygons are referenced in manual mode, which increases the load on the geometry engine (and reduces rendering efficiency). Because of this, be careful when using Manual sort mode when there are a large number of translucent polygons.

Opaque polygons are always Auto sorted.

Regardless of the sort mode, opaque polygons are always rendered before translucent polygons.

Note: Maximum Polygon Y value on the LCD

The Y coordinates are the inverse of Y coordinates in the BG Screen Coordinate Group in the Coordinate Transformation Flow Chart (Figure 7-3). Therefore, the maximum Y value for a polygon on the LCD is the minimum value in the BG Screen Coordinate Group.

- Reflecting the Depth Buffering Select Flag and Translucent Polygon Y Sorting Select Flag

  These flags are reflected in the geometry engine from the next frame. However, because polygon list RAM and vertex RAM are double buffered, the rendering engine renders the data that the geometry engine stored in the previous frame. Therefore, data that the geometry engine outputs is rendered with an additional one-frame delay.

- Processing the SwapBuffers command

  The SwapBuffers command is processed at the next V-Blank, regardless of when it was input. (The geometry engine is in wait status until the V-Blank period arrives.) The Polygon List RAM, Vertex RAM, rendering-related registers, and other data referenced by the Rendering Engine is swapped at the start of the next V-Blank period after the issuance of the SwapBuffers command. Because of this timing, rendering reflects the written graphics data in the next frame after the SwapBuffers command is issued.

7.2.8 Viewport

ViewPort: Sets the Viewport

Name: VIEWPORT Address: 0x04000580 Attribute: W Command Code: 0x60

31 24	23 16	15 8	7 0
INTEGER_Y2	INTEGER_X2	INTEGER_Y1	INTEGER_X1
Y2	X2	Y1	X1

- Y2, X2 [d31–d24, d23–d16] : Top right coordinates

  Set Y2 to a value larger than Y1. Can be set in the range 0–191.

  Set X2 to a value larger than X1. Can be set in the range 0–255.

- Y1, X1[d15–d08, d07–d00] : Bottom left coordinates (the viewport origin)

  Set Y1 to a value smaller than Y2. Can be set in the range 0–191.

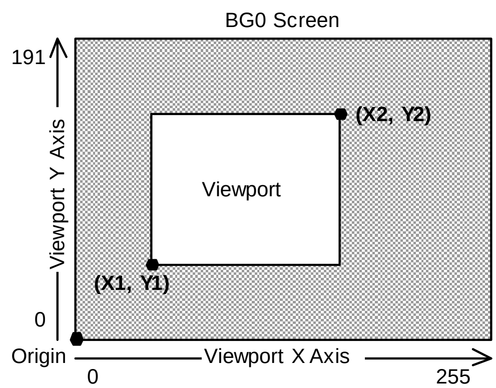
  Set X1 to a value smaller than X2. Can be set in the range 0–255.

This sets the position and the size of the viewport which draws 3D graphics on the BG0 screen.

The BG0's H offset is added to get the display position on the LCD.

Notice that the origin point is different than for the 2D coordinate group. (See Figure 7-14)

Figure 7-14 : Size and Position of the Viewport



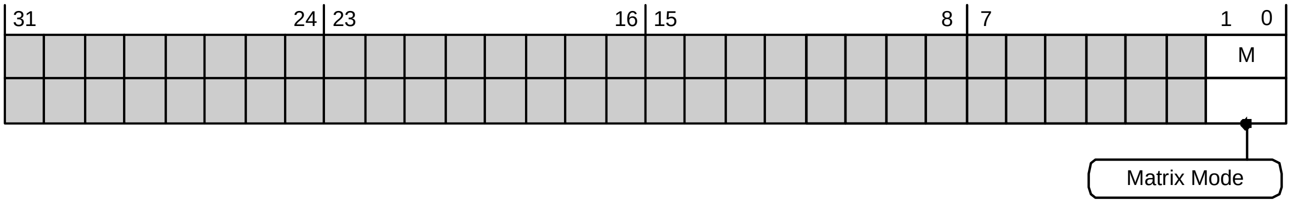
Note: Rendering may result in one dot protruding from the right or bottom edge of the viewport.

7.2.9 Matrices

7.2.9.1 Manipulating the Current Matrix

MatrixMode: Sets the Matrix Mode

Name: MTX_MODE Address: 0x04000440 Attribute: W Command Code: 0x10



- M[d01–d00] : Matrix mode

00	Projection mode	... (For manipulating projection matrices)
01	Position mode	... (For manipulating position coordinate matrices)
10	Position & Vector Simultaneous Set mode	... (For manipulating position coordinate matrices and directional vector matrices)
11	Texture mode	... (For manipulating texture matrices)

This specifies the current matrix on which Matrix commands operate (this classification is called the matrix mode).

Position mode and Position and Vector Simultaneous Set mode

TWL does not use the hardware to make a unit normal vector. Therefore, to obtain correct lighting effects, you must set a unit vector as the normal in advance, and the directional vector matrix must be an *orthogonal matrix* (a matrix that does not change the length of the directional vector). When a model is transformed, the transformation matrix is usually used for both the position coordinates matrix and the directional vector matrix. But with TWL, the directional vector matrix must be maintained as an orthogonal matrix, so depending on the type of transformation, sometimes the transformation matrix is used only for the position coordinates matrix.

Let the situation dictate whether to use Position mode or Position and Vector Simultaneous Set mode to set the Position and Vector simultaneously.

Examples

When rotating a model, usually the mode is set to the Position and Vector Simultaneous Setting and the MultMatrix command is executed to apply the rotation component of the matrix for both the position coordinates matrix and the directional vector matrix.

When the Scale command is used for model scaling, the directional vector matrix can be maintained as an orthogonal matrix, so the correct lighting effect can be obtained. (See the Scale command on page 210.)

When the scale matrix is applied with the MultMatrix command, the directional vector matrix is not maintained as an orthogonal matrix, and the lighting effect is brighter or darker than the original. Thus, it would be safer not to use this procedure.

When Position mode is selected, Matrix commands are applied only to the position coordinates matrix, so the correct lighting effect is obtained even when the scale matrix is applied with the MultMatrix command. You can also use this mode when you want to apply the rotation matrix only to the position coordinates matrix, but unnatural effects can arise. For example, sometimes the lighting effect does not change upon rotation, or a part that is not being illuminated ends up being the brightest.

MTX_IDENTITY: Initialize Current Matrix to Unit Matrix

Name: MTX_IDENTITY Address: 0x04000454 Attribute: W Command Code: 0x15

31 – 24	23 – 16	15 – 8	7 – 0

LoadMatrix44: Set 4x4 Matrix to Current Matrix

Name: MTX_LOAD_4x4 Address: 0x04000458 Attribute: W Command Code: 0x16

31	30 – 12	11 – 0
S	INTEGER_M44	DECIMAL_M44
Sign	Integer part	Decimal part

Signed fixed-point number (sign + 19-bit integer + 12-bit fractional part)

- M44 : 4x4 matrix elements m[x] (x = 0 - 15)

The matrix M is set as follows with elements m[0] to m[15]:

$$M = \begin{bmatrix} m[0] & m[1] & m[2] & m[3] \\ m[4] & m[5] & m[6] & m[7] \\ m[8] & m[9] & m[10] & m[11] \\ m[12] & m[13] & m[14] & m[15] \end{bmatrix}$$

LoadMatrix43: Set 4x3 Matrix to Current Matrix

Name: MTX_LOAD_4x3 Address: 0x0400045C Attribute: W Command Code: 0x17

31	30 – 12	11 – 0
S	INTEGER_M43	DECIMAL_M43
Sign	Integer part	Decimal part

Signed fixed-point number (sign + 19-bit integer + 12-bit fractional part)

- M43 : 4x3 matrix elements m[x] (x = 0 - 11)

The matrix M is set as follows with elements m[0] to m[11]:

$$M = \begin{bmatrix} m[0] & m[1] & m[2] & 0 \\ m[3] & m[4] & m[5] & 0 \\ m[6] & m[7] & m[8] & 0 \\ m[9] & m[10] & m[11] & 1 \end{bmatrix}$$

MultMatrix44: Multiply 4x4 Matrix by Current Matrix

Name: MTX_MULT_4x4 Address: 0x04000460 Attribute: W Command Code: 0x18

31	30 … 24 | 23 … 16 | 15 … 12	11 … 8 | 7 … 0
S	INTEGER_M44	DECIMAL_M44
Sign	Integer part	Decimal part

Signed fixed-point number (sign + 19-bit integer + 12-bit fractional part)

- M44 : 4x4 matrix elements m[x] (x = 0 - 15)

The matrix M is set as follows with elements m[0] to m[15]:

$$M = \begin{bmatrix} m[0] & m[1] & m[2] & m[3] \\ m[4] & m[5] & m[6] & m[7] \\ m[8] & m[9] & m[10] & m[11] \\ m[12] & m[13] & m[14] & m[15] \end{bmatrix}$$

If the current matrix is C, then the new current matrix C' = MC

MultMatrix43: Multiply 4x3 Matrix by Current Matrix

Name: MTX_MULT_4x3 Address: 0x04000464 Attribute: W Command Code: 0x19

31	30 … 24 | 23 … 16 | 15 … 12	11 … 8 | 7 … 0
S	INTEGER_M43	DECIMAL_M43
Sign	Integer part	Decimal part

Signed fixed-point number (sign + 19-bit integer + 12-bit fractional part)

- M43 : 4x3 matrix elements m[x] (x = 0 - 11)

The matrix M is set as follows with elements m[0] to m[11]:

$$M = \begin{bmatrix} m[0] & m[1] & m[2] & 0 \\ m[3] & m[4] & m[5] & 0 \\ m[6] & m[7] & m[8] & 0 \\ m[9] & m[10] & m[11] & 1 \end{bmatrix}$$

If the current matrix is C, then the new current matrix C' = MC

MultMatrix33: Multiply 3x3 Matrix by Current Matrix

Name: MTX_MULT_3x3 Address: 0x04000468 Attribute: W Command Code: 0x1A

31	30 ... 24 | 23 ... 16 | 15 ... 12	11 ... 8 | 7 ... 0
S	INTEGER_M33	DECIMAL_M33
Sign	Integer part	Decimal part

Signed fixed-point number (sign + 19-bit integer + 12-bit fractional part)

- M33 : 3x3 matrix elements m[x] (x = 0 - 8)

The matrix M is set as follows with elements m[0] to m[8]:

$$M = \begin{bmatrix} m[0] & m[1] & m[2] & 0 \\ m[3] & m[4] & m[5] & 0 \\ m[6] & m[7] & m[8] & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

If the current matrix is C, then the new current matrix C’ = MC

Translate: Multiply Translation Matrix by Current Matrix

Name: MTX_TRANS Address: 0x04000470 Attribute: W Command Code: 0x1C

31	30 ... 24 | 23 ... 16 | 15 ... 12	11 ... 8 | 7 ... 0
S	INTEGER_TRANSLATE	DECIMAL_TRANSLATE
Sign	Integer part	Decimal part

Signed fixed-point number (sign + 19-bit integer + 12-bit fractional part)

- TRANSLATE : Translation matrix elements m[x] (x = 0 - 2)

The matrix M is set as follows with elements m[0] to m[2]:

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ m[0] & m[1] & m[2] & 1 \end{bmatrix}$$

If the current matrix is C, then the new current matrix C’ = MC

Scale: Multiply the Scale Matrix by Current Matrix

Name: MTX_SCALE Address: 0x0400046C Attribute: W Command Code: 0x1B

31	30 – 12	11 – 0
S	INTEGER_SCALE	DECIMAL_SCALE
Sign	Integer part	Decimal part

Signed fixed-point number (sign + 19-bit integer + 12-bit fractional part)

- SCALE : Scale matrix elements m[x] (x = 0 - 2)

The matrix M is set as follows with elements m[0] to m[2]:

$$M = \begin{bmatrix} m[0] & 0 & 0 & 0 \\ 0 & m[1] & 0 & 0 \\ 0 & 0 & m[2] & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

If the current matrix is C, then the new current matrix C’ = MC

The Scale command performs multiplication only on the position coordinates matrix, even when the matrix mode has been set to Position & Vector Simultaneous Setting mode. (If it were performed on the directional vector matrix, the direction and length of vectors would change and abnormal lighting effects would arise.)

7.2.9.2 Matrix Stack

Manipulations are made to the stack of the current matrix.

Because the position coordination matrix stack and the directional vector matrix stack are connected, manipulations affect both stacks, whether the mode is set to Position mode or Position & Vector Simultaneous Setting mode.

PushMatrix: Push the Current Matrix on the Stack

Name: MTX_PUSH Address: 0x04000444 Attribute: W Command Code: 0x11

31–24	23–16	15–8	7–0

PopMatrix: Pop the Current Matrix from the Stack

Name: MTX_POP Address: 0x04000448 Attribute: W Command Code: 0x12

31–24	23–16	15–8	7–6	5	4–0
				S	INT
				NUM number of pops	

Signed integer (sign + 5-bit integer)

- NUM[d05–d00] : Specify the number of pops (Can set a value of -30 to 31)

Pops the n^{th} level matrix (as specified by NUM), starting from the position of the stack pointer of the matrix stack specified by the matrix mode, and sets it as the current matrix.

When the matrix mode is set to projection mode, the stack only has 1 level so the value of NUM is treated as 1, no matter what value has been set.

This command is normally issued outside of the command string that runs from Begin to End, but it can also be issued between Vertex commands within the Begin to End command string.

Command String Example 1	PushMatrix→Translate→Begin→Vertex→Vertex→Vertex→End→PopMatrix (1)
Command String Example 2	PushMatrix→Translate→PushMatrix→Begin→Vertex→PopMatrix (1)→Vertex→ PopMatrix (-1)→Vertex→PopMatrix (1)→Vertex→End→PopMatrix (1)

In Command string Example 2, the PopMatrix command is issued between Vertex commands to realize stitching and sprite polygon deformations. Stitching is a type of skinning. Sprite polygons are polygons displayed in 2D.

StoreMatrix: Store the Current Matrix in the Specified Stack Position

Name: MTX_STORE Address: 0x0400044C Attribute: W Command Code: 0x13

31–5	4–0
	INDEX
	Storage Position

Unsigned integer (5-bit integer)

- INDEX[d04–d00] : Storage position (Can set a value of 0 to 30)

Stores the matrix specified with matrix mode in the matrix stack in the position specified by INDEX.

When the matrix mode is set to projection mode, the stack only has 1 level so the value of INDEX is treated as 0 no matter what value has been set.

The matrix stack pointer moved by the PushMatrix and PopMatrix commands does not move after this command is issued.

RestoreMatrix: Reads Matrix from Specified Position in Stack

Name: MTX_RESTORE Address: 0x04000450 Attribute: W Command Code: 0x14

31–5	4–0
	INDEX
	Read Position

Unsigned integer (5-bit integer)

- INDEX[d04–d00] : Read position (Can set a value of 0 to 30)

The value in the position specified by INDEX in the matrix stack is set as the matrix specified by the matrix mode.

When the matrix mode is set to projection mode, the stack only has 1 level so the value of INDEX is treated as 0 no matter what value has been set.

The matrix stack pointer moved by the PushMatrix and PopMatrix commands does not move after this command is issued.

This command is normally issued outside of the command string that runs from Begin to End, but it can also be issued between Vertex commands within the Begin to End command string.

Command String Example 1	StoreMatrix (i)⟶Translate⟶Begin⟶Vertex⟶Vertex⟶Vertex⟶End⟶ RestoreMatrix (i)
Command String Example 2	StoreMatrix (i)⟶Translate⟶StoreMatrix (i+1)⟶Begin⟶Vertex⟶RestoreMatrix (i)⟶ Vertex⟶RestoreMatrix (i+1)⟶Vertex⟶End⟶RestoreMatrix (i)

In Command string Example 2, the RestoreMatrix command is issued between Vertex commands to realize stitching and sprite polygon deformations. Stitching is a type of skinning. Sprite polygons are polygons displayed in 2D.

7.2.9.3 Reading the Current Matrix

ClipMatrix_Result: Read the Current Clip Coordinates Matrix

Name	Address	Attribute	Initial Value
CLIPMTX_RESULT_x (x=0-15)	0x04000640, 0x04000644, 0x04000648, 0x0400064C, 0x04000650, 0x04000654, 0x04000658, 0x0400065C, 0x04000660, 0x04000664, 0x04000668, 0x0400066C, 0x04000670, 0x04000674, 0x04000678, 0x0400067C	R	0x00000000

31	30–12	11–0
S	INTEGER_m[x]	DECIMAL_m[x]
Sign	Integer part	Decimal part

Signed fixed-point number (sign + 19-bit integer + 12-bit fractional part)

- m[x] (x = 0–15) : The elements of the current clip coordinates matrix

$$CurrentClipCoordinatesMatrix = \begin{bmatrix} m[0] & m[1] & m[2] & m[3] \\ m[4] & m[5] & m[6] & m[7] \\ m[8] & m[9] & m[10] & m[11] \\ m[12] & m[13] & m[14] & m[15] \end{bmatrix}$$

The current clip coordinates matrix (position coordinate matrix and projection matrix) can be read.

If you want to read the current projection matrix, make the current position coordinates matrix a unit matrix and read this register.

If you want to read the current position coordinates matrix, make the current projection matrix a unit matrix and read this register.

Note: To safely read these matrices, confirmation that the Geometry Engine is stopped must occur before reading.

VectorMatrix: Read the Current Directional Vector Matrix

Name	Address	Attribute	Initial Value
VECMTX_RESULT_x (x =0-8)	0x04000680, 0x04000684, 0x04000688, 0x0400068C, 0x04000690, 0x04000694, 0x04000698, 0x0400069C, 0x040006A0	R	0x00000000

31	30–12	11–0
S	INTEGER_m[x]	DECIMAL_m[x]
Sign	Integer part	Decimal part

Signed fixed-point number (sign + 19-bit integer + 12-bit fractional part)

- m[x] (x = 0 - 8) : The elements of the current directional vector matrix

$$CurrentDirectionalVectorMatrix = \begin{bmatrix} m[0] & m[1] & m[2] \\ m[3] & m[4] & m[5] \\ m[6] & m[7] & m[8] \end{bmatrix}$$

Note: To safely read this matrix, first confirm that the Geometry Engine is stopped.

TWL-06-0017-001-D
Released: February 16, 2009

7.2.10 Light

TWL supports only parallel light sources.

LightVector: Set the Light's Directional Vector

Name: LIGHT_VECTOR Address: 0x040004C8 Attribute: W Command Code: 0x32

31 30	29	28 24 23 20	19	18 16 15 10	9	8 7 0
LNUM	SZ	DECIMAL_Z	SY	DECIMAL_Y	SX	DECIMAL_X
Light	Directional vector's Z component		Directional vector's Y component		Directional vector's X component	

Signed fixed-point number (sign + 9-bit fractional part)

- LNUM[d31–d30] : Light number

  0–3

- X, Y, Z[d29–d20], [d19–d10], [d09–d00] : Directional vector

  Coordinate transformation with the directional vector matrix is performed after the settings are made.

  The hardware does not perform vector normalization, so set the unit vector.

LightColor: Set the Light Color

Name: LIGHT_COLOR Address: 0x040004CC Attribute: W Command Code: 0x33

31 30	29 24 23 16 15	14 10	9 8 7 5	4 0
LNUM		BLUE	GREEN	RED
Light		Light Color		

- LNUM[d31–d30] : Light number

  0–3

- [d14–d00] : Light color

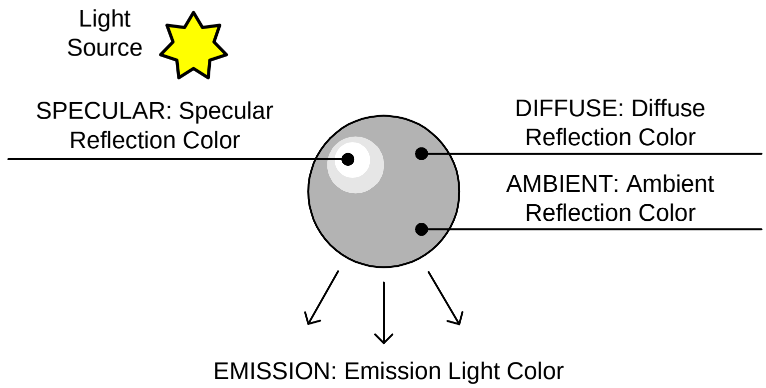
  Although OpenGL has light color parameters for diffuse, specular, and ambient light, for TWL this has been simplified to a single parameter.

7.2.11 Material

For objects, the appearance of the texture differs depending on the material on the surface of the object and the environment in which the object sits.

As shown in Figure 7-15, lighting (the illumination process) uses four material colors (specular reflection color, diffuse reflection color, ambient reflection color, and emission light color) to express the texture of a model.

Figure 7-15 : Material Color Schematic



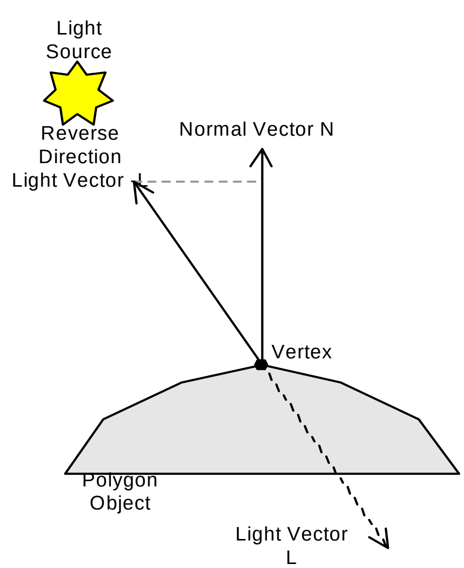
Diffuse reflection color

This is the color of the object when it is illuminated by the light. Consider this the basic color of the object.

The diffuse reflection color is defined to reflect evenly in all directions, so it is not influenced by the position of the eye point. As shown in Figure 7-16, it is, however, influenced by the color and the direction of the light and by the normal of the polygon.

The only influence it has is on the color of the parts of objects that are directly illuminated by the light.

Figure 7-16 : Directional Vector Relational Diagram (Diffuse Reflection Color)



Normal vector N: The unit normal vector of the vertex. (Set with the Normal command)

Light vector L: Normalized vector of parallel light source. (Set with the LightVector command)

Ambient reflection color

This is the color of the object when it is illuminated by ambient light.

Objects are illuminated not only by direct light, but also by light reflecting off of other objects.

This reflected light is called ambient light when it is defined to exist uniformly in the entire scene. Since ambient light exists uniformly in the entire scene, it influences the color of the entire object.

Diffuse reflection color has strong influence of parts of the object that are illuminated by direct light, but ambient reflection color has the predominant influence on parts that are shaded.

Specular reflection color

This is the glossy color of the object when it is illuminated by light. This glossiness is called specular highlight.

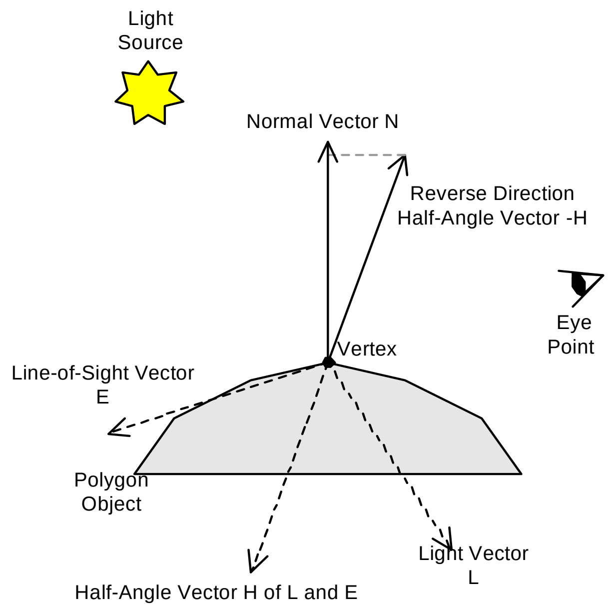
In optical terms, specular highlight is the reflected light of the light source. Accordingly, the part of the object where light strikes and reflects straight back to the eye point is the brightest.

Specular highlight is influenced by the color and direction of light, the normal of the polygon and the position of the eye point. (See Figure 7-17.)

When the eye point shifts the specular highlight moves.

It only has influence on the color of the parts of objects that are directly illuminated by the light, and this influence corresponds to the eye point.

Figure 7-17 : Directional Vector Relational Diagram (Specular Reflection Color)



Normal vector N: The unit normal vector for the vertex. (Set with the Normal command.)

Line-of-sight vector E: Normalized vector from eye point toward vertex. Taken to be the same as the negative direction of the z axis in the View coordinate system.

Light vector L: Normalized vector of parallel light source. (Set with the LightVector command.)

Half-angle vector H of L and E: Normalized vector of the sum of the line-of-sight vector and the light vector.

When TWL performs the calculation for specular reflection shininess, the line-of-sight vector is taken to be the same as the negative direction of the z axis in the View coordinate system, and it is assumed that both the light vector and the normal vector will be transformed into the View coordinate system.

For this reason, when the View matrix (the LookAt matrix) is applied to the projection matrix, the coordinate system for the light vector and normal vector differ from the coordinate system for the line-of-sight vector after the transformation, and the specular reflection result is abnormal.

Accordingly, when the specular reflection color is set to any value other than black (0), the matrix mode must be set to the Position & Vector Simultaneous Set mode, and the rotation component of the view matrix (the LookAt matrix) must be reflected on the directional vector matrix.

When the specular reflection color is set to black, the view matrix can be applied to the projection matrix because diffusion reflection does not depend on the eye point. In short, you can set the model matrix for the position coordinates matrix, and the combination of the view matrix and the projection matrix for the projection matrix.

Emission light color

This is the color of the light that is emitted from the object itself.

Note that this is not treated as light, so it does not illuminate other objects (that is, it does not influence the color of other objects).

To achieve this result, you need to create a light source that is the same color as the emission light color and place it at the same position as the object that you want emitting light.

MaterialColor0: Set the Material's Diffuse Reflection Color and Ambient Reflection Color

Name: DIF_AMB　　Address: 0x040004C0　　Attribute: W　　Command Code: 0x30

31	30 … 26	25 24 23 … 21	20 … 16	15	14 … 10	9 8 7 … 5	4 … 0
	AMBIENT_BLUE	AMBIENT_GREEN	AMBIENT_RED	C	DIFFUSE_BLUE	DIFFUSE_GREEN	DIFFUSE_RED
	Ambient reflection color			Vertex	Diffuse reflection color		

- C[d15]: Vertex color set flag

0	Does not set vertex color
1	Sets the diffuse reflection color as the vertex color

If diffuse reflection color has been set for the vertex color, it remains valid until the next time the Color, Normal or MaterialColor0 (vertex color set flag) command is issued and the current vertex color is updated.

Because the vertex color is handled as bits R:G:B = 6:6:6 in the Rendering Engine, the diffuse reflection color is applied to the upper five bits. When diffuse reflection color is 0, the lower 1 bit is 0, and when the diffuse reflection color is nonzero, the lower 1 bit is 1.

MaterialColor1: Set the Material's Specular Reflection Color and Emission Color

Name: SPE_EMI　　Address: 0x040004C4　　Attribute: W　　Command Code: 0x31

31	30 … 26	25 24 23 … 21	20 … 16	15	14 … 10	9 8 7 … 5	4 … 0
	EMISSION_BLUE	EMISSION_GREEN	EMISSION_RED	S	SPECULAR_BLUE	SPECULAR_GREEN	SPECULAR_RED
	Emission color			Shininess	Specular reflection color		

- S[d15] : Specular reflection shininess table - enable flag

0	Disable
1	Enable

Shininess: Set the Specular Reflection Shininess Table

Name: SHININESS Address: 0x0400004D Attribute: W Command Code: 0x34

31 24	23 16	15 8	7 0
SHININESS_n (n=4x+3)	SHININESS_n (n=4x+2)	SHININESS_n (n=4x+1)	SHININESS_n (n=4x+0)
Shininess when Is=n	Shininess when Is=n	Shininess when Is=n	Shininess when Is=n

Unsigned fixed point decimal (8-bit fractional part)

Sets the 8-bit x 128 table for converting the shininess of the specular reflection.

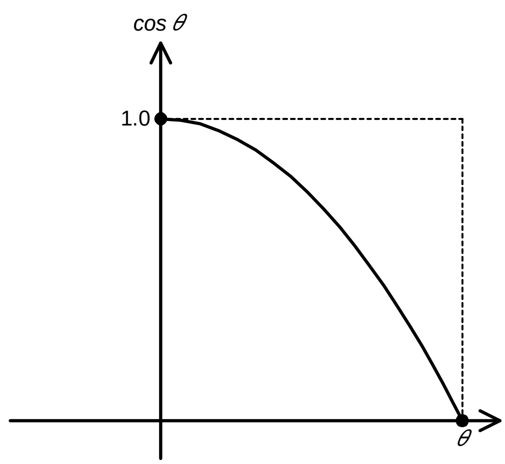
If the specular reflection shininess table-enable flag was set to 1 by the just-issued MaterialColor1 command, the Geometry Engine looks up the table based on the upper 7 bits of *Is*—the result of the specular calculation—and converts the shininess of the specular reflection.

This table can be used to adjust the brightness of the specular reflection. (See "7.2.11.1 Lighting (Illumination Process)" on page 219 for the computation formula.)

By rewriting the specular reflection shininess table, you can display polygons having a number of different specular reflection effects inside a single scene.

The specular reflection shininess calculation result *Is* is obtained from the inner product of the vectors, so as Figure 7-18 shows, it is less precise near the center of the luster and more precise farther away (A·B = |A| |B|cosθ). (See "7.2.11.1 Lighting (Illumination Process)" on page 219 for the *Is* calculation.)

Figure 7-18 : Specular Reflection Shininess



Technique

You can achieve special lighting effects by setting non-consecutive values for the specular reflection shininess table.

7.2.11.1 Lighting (Illumination Process)

Lighting (the illumination process) is conducted when the Normal command is issued, and the results of the calculations are used for the vertex color.

The computations shown below are done on each color component (R, G, B).

Lighting Formulas for Various Material Colors

Material Color	Lighting Formulas
Diffuse Reflection Color	ld = max[0, –L•N] D = ld*light*diffuse_material
Ambient Reflection Color	A = light*ambient_material
Specular Reflection Color	ls = max[0, cos 2 θ] (When the specular reflection shininess table is disabled) S = ls*light*specular_material (When the specular reflection shininess table is enabled) S = shininess_table[ls]*light*specular_material
Emission Color	E = emission_material

L : The light's directional vector

N : Normal vector

H : The vector that is half the sum of L (the light's directional vector) and the line-of-sight vector (the vector that points in the negative direction of the Z axis.) This is called a *half-angle vector* because it indicates the direction halfway between the L and line-of-sight vectors.

θ : The angle between the vector (-H) and the vector (N)

ld : Diffuse reflection shininess

ls : Specular reflection shininess

light : Light color

diffuse_material : Material's diffuse reflection color

ambient_material : Material's ambient reflection color

specular_material : Material's specular reflection color

emission_material : Material's emission color

Vertex color expressions

The ultimate vertex color is calculated with the following expression using the results of the lighting calculations conducted on each material color.

$$C = \sum_{i=0}^{3} [Di + Ai + Si] + E$$

C: Vertex color

Di : Diffuse reflection color for light *i*

Ai : Ambient reflection color for light *i*

Si : Specular reflection color for light *i*

E : The color of self-emitted light

When light *i* is disabled, the corresponding color components (Di, Ai, Si) are not calculated.

The greater the number of lights that are enabled, the greater the load of the vertex color computations (that is, the greater the load of the Normal command). For this reason, be careful not to enable any more lights than are needed.

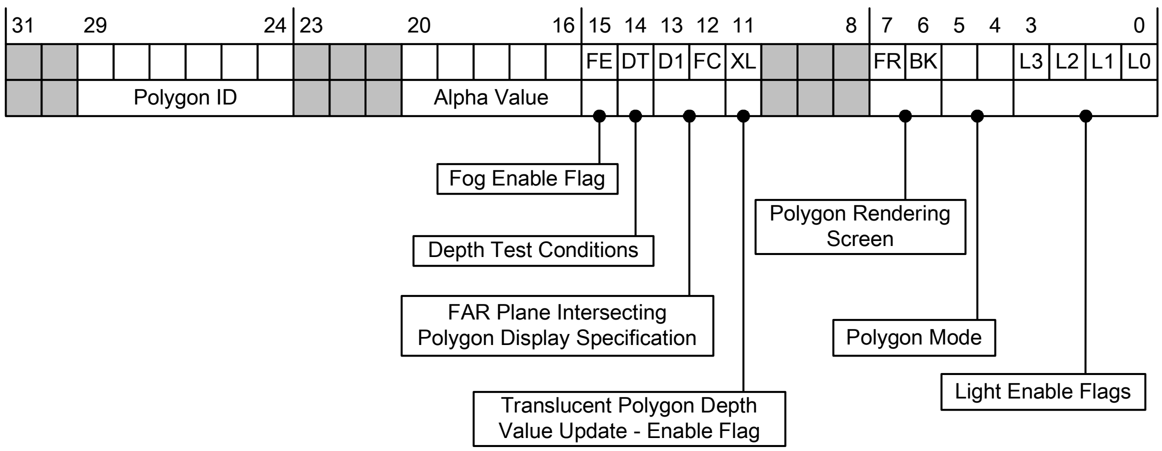
Vertex color when lighting is OFF

Even when lighting is OFF, the vertex color is calculated using the above expressions when the Normal command is issued. The result in this case is that the vertex color is set to the emission color.

7.2.12 Polygon Attributes

PolygonAttr: Set the Polygon-Related Attribute Values

Name: POLYGON_ATTR Address: 0x040004A4 Attribute: W Command Code: 0x29



- [d29–d24] : Polygon ID

  The polygon ID is stored in separate attribute buffers for opaque polygons and translucent polygons when the polygon is being rendered by the Rendering Engine. The stored polygon ID is used when rendering translucent polygons and shadow polygons and when edge-marking. For details, see "7.3.4 Rasterizing" on page 260.

- [d20–d16] : alpha value

1–31	Polygon's opaqueness
0	Wire frame display

The polygon is called a *translucent polygon* when $1 \leq \alpha \leq 30$ and an *opaque polygon* when $\alpha = 31$. When $\alpha = 0$, the display becomes a wireframe display and the original meaning of α is lost.

- FE[d15] : Fog-enable flag

  When fog is enabled, the Rendering Engine performs fog blending.

  To learn about fog blending, see "7.3.9 Fog Blending" on page 285.

0	Disable
1	Enable

- DT[d14] : Depth test conditions

  When set to 1, another polygon can be pasted on a polygon that has already been rendered (decal polygon).

0	Rendering when the fragment's depth value is smaller than the depth buffer's depth value.
1	Rendering when the fragment's depth value is equal to the depth buffer's depth value.

- [d13–d12] : Polygon display specification
  - D1[d13] : 1-dot polygon rendering specification

    This allows control of whether 1-dot polygons are passed to the rendering engine. A 1-dot polygon is a polygon where the coordinates (x, y) of all the vertices are integrated into a single coordinate as the results of geometry engine calculations.

0	Does not render if becomes a 1-dot polygon
1	Renders even if becomes 1-dot polygon

    When set to 1, 1-dot polygons are always written to polygon list RAM and vertex RAM.

    When this is set to 0, the depth value of the 1-dot polygon controls whether to write the polygon to polygon list RAM and vertex RAM or to discard it.

    Set the display boundary depth value of 1-dot polygons with the Disp1DotDepth register.

  - FC[d12] : Far plane intersecting polygon display specification

0	Deletes if intersects the far plane
1	Clips if intersects the far plane

    Note that clipping on the far plane increases the load on the Geometry Engine.

- XL[d11] : Translucent polygon depth-value update enable flag

  Select whether to update the depth buffer when rendering a polygon with an α value of 1–30.

  When this is set to 1, sometimes you can improve on a situation where too much fog is applied in regions where translucent polygons are being rendered. However, you need to be careful because sometimes the edge-marking of background is not rendered correctly.

0	Does not update the depth buffer when rendering translucent polygons
1	Updates the depth buffer when rendering translucent polygons

- [d07–d06] : Polygon rendering screen specification

  The surface is the plane tracing the vertices counterclockwise.

  - FR[d07] : Render front surface

0	Disable
1	Enable

  - BK[d06] : Render back surface

0	Disable
1	Enable

  If the specified screen is in the screen being displayed when the rendering specification is disabled, this polygon will not be included in the List RAM.

For quadrilateral polygons, if any of the first three vertices share the same coordinates, duplication will be detected by the hardware and the polygon is displayed as usual without regard to front or back. Furthermore, when the first three vertices do not overlap but are in a straight line, the straight line is not preserved due to a problem with precision in internal calculations. In this case, the surface may be determined to be the front or back according to the camera state. Use the following procedure to avoid this problem:

- Change the order the vertices are sent
- Have the second coordinate value be the same as the first or third coordinate value (Of the first three vertices, set 1 and 2 or 2 and 3 to the same value)
- Separate into triangles

When rendering a line segment in which polygon vertex coordinates overlap, front/back determination is impossible, and therefore it is always rendered, regardless of this flag's setting.

- PM[d05–d04] : Polygon mode

  Modulation mode and decal mode are ways of blending texture color and fragment color.

  Toon / Highlight shading is a way to transform with the fragment color table.

  Shadow polygon is a feature for applying shadow using the stencil buffer.

  For details, see the respective parts in the Rendering Engine "7.3.1 Overview" on page 251.

00	Modulation mode
01	Decal mode
10	Toon / Highlight shading
11	Shadow polygon

About Toon/Highlight Shading

Toon and Highlight Shading share use of the same table.

Use the DISP3DCN register to choose either Toon or Highlight.

The setting written to the DISP3DCN register becomes valid when the frame switches, so Toon and Highlight Shading cannot be mixed in the same drawing frame.

- L3–L0[d03–d00] : Light enable flags

  These are separate flags for setting lights 0–3.

0	Disable (light off)
1	Enable (light on)

Where to issue the PolygonAttr command

The values set with the PolygonAttr command become valid when the Begin command is issued. The values are subsequently used as vertex attributes.

Do not issue the PolygonAttr command between the Begin command and the End command.

Simply setting the light enable flag to enabled in the Begin command does not affect the vertex color.

The vertex color is first affected when lighting (lighting process) is performed with the Normal command after the light enable flag setting is enabled in the Begin command.

Disp1DotDepth: 1-Dot Polygon Display Boundary Depth Value Register

Name: DISP_1DOT_DEptH Address: 0x04000610 Attribute: W Initial value: 0x7FFF

15	14	8	7	0
	INTEGER_W			DECIMAL_W
	W Coordinate			

Fixed-point number (12-bit integer + 3-bit fractional part)

- W coordinates [d14–d00] : Depth value

  When the PolygonAttr command’s “1-dot polygon rendering specification flag” is 0, the Geometry Engine references this register for use as described below:

  When the X and Y coordinates of all polygon vertices are transformed into BG screen coordinates within a range of 1 dot or less, if the smallest W value (the depth value) is larger than this register’s setting value, polygon data is not written to Polygon List RAM or Vertex RAM (and is not displayed as a result).

  This W value is referenced even during Z buffering.

7.2.13 Polygons

BEGIN_VTXS: Declare the Start of the Vertex List

Name: BEGIN_VTXS   Address: 0x04000500   Attribute: W   Command Code: 0x40

31–24	23–16	15–8	7–2	1–0
				TYPE
				Type

- TYPE[d01–d00] : Primitive type

00	Triangle
01	Quadrilateral
10	Triangle Strips
11	Quadrilateral Strips

Polygon strips share vertices, so they consume less Vertex RAM than independent polygons of the same shape.

The table below shows the relation between drawing and the order in which vertices are issued by the Vertex command for different types of primitives.

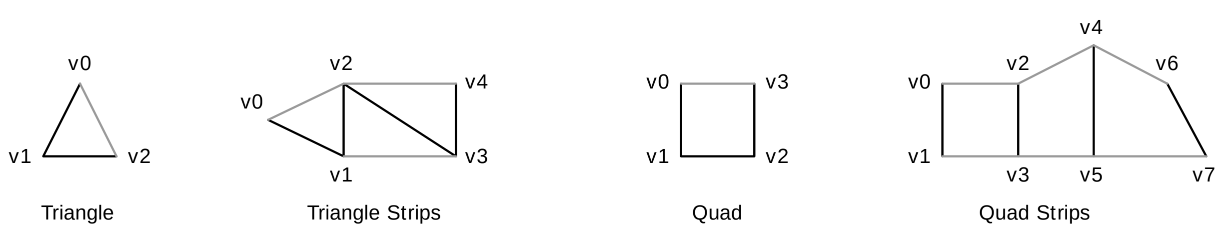
Relation Between Drawing and the Order in Which the Vertex Command Issues Vertices

Triangle Polygon	A series of triangles are drawn starting with vertices “v0, v1, and v2” and then vertices “v3, v4, and v5.”
Quadrilateral Polygon	A series of quadrilaterals are drawn starting with vertices “v0, v1, v2, and v3” and then vertices “v4, v5, v6, and v7.”
Triangle Strips	A series of triangles are draw starting with vertices “v0, v1, and v2,” “v2, v1, and v3” and then “v2, v3, and v4.” This is the order so that the triangles are drawn in the same direction on both sides of the surface. (See Figure 7-15.)
Quadrilateral Strips	A series of quadrilaterals are drawn starting with vertices “v0, v1, v3, and v2,” “v2, v3, v5, and v4” and then “v4, v5, v7, and v6.” This is the order so that the quadrilaterals are drawn in the same direction on both sides of the surface. (See Figure 7-15.)

Defining the primitive's front surface

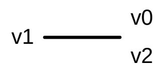
The surface is described counterclockwise (v0, v1, v2,... : in order of issued Vertex-group commands).

Figure 7-19 : Order in Which the Vertex Commands Issues Vertices



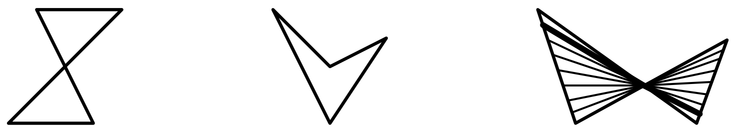
When you want to draw line segments, set the same value for neighboring vertices in the above primitives. However, because it is not possible to determine the front or back of the line segments, they are always rendered, even if the polygon attributes specify to disable display. As Figure 7-20 shows, the anti-aliasing and edge marking features (see the Rendering Engine "7.3.1 Overview" on page 251) work on line segments as well.

Figure 7-20: Line Segment Using Sides from a Triangle



When you draw quadrilateral polygons in the shapes shown in Figure 7-21, sometimes the results are not as intended. Be sure to set the vertices so quadrilateral polygons are not drawn in these shapes.

Figure 7-21: Quadrilateral Polygon Shapes that Yield Unintended Shapes



Quadrilateral with Intersecting Sides — Concave Quadrilateral — Twisted Quadrilateral

End: Declare the End of the Vertex List

Name: END_VTXS Address: 0x04000504 Attribute: W Command Code: 0x41

31–24	23–16	15–8	7–0

Make certain to issue Begin and End commands in pairs.

Color: Directly Set the Vertex Color

Name: COLOR Address: 0x04000480 Attribute: W Command Code: 0x20

31–24	23–16	15	14–10	9–5	4–0
			BLUE	GREEN	RED
			Color		

The vertex color remains valid until the current vertex color is updated by the next Color command, Normal command, or MaterialColor0 (vertex color set flag) command.

Thus, multiple vertices can share the same vertex color.

Because the vertex color is handled as bits R:G:B = 6:6:6 in the Rendering Engine, the diffuse reflection color is applied to the upper five bits. When set color value is 0, the lower 1 bit is 0, and when it is nonzero, the lower 1 bit is 1.

The Color command is usually issued between the Begin and End commands, but it can also be issued before the Begin command.

Command string Example 1	Begin→Color→Vertex→Vertex→Vertex→End
Command string Example 2	Color→Begin→Vertex→Vertex→Vertex→End→Begin→Vertex→Vertex→Vertex→End

Normal: Set the Normal Vector

Name: NORMAL Address: 0x04000484 Attribute: W Command Code: 0x21

31	30	29	28–20	19	18–10	9	8–0
		S	NZ	S	NY	S	NX
		Normal vector’s Z component		Normal vector’s Y component		Normal vector’s X component	

Signed fixed-point number (sign + 9-bit fractional part)

Lighting (illumination process) is performed only when the Normal command is executed.

Accordingly, you need to reissue the Normal command after you switch lighting On/Off or change the light or material parameters in order for the change to be reflected in the vertex color. (See note.)

Further, the hardware does not normalize vectors, so you need to set the unit vector.

The vertex color obtained with the lighting process remains valid until the current vertex color is updated by the next Color command, Normal command, or MaterialColor0 (vertex color set flag) command.

Thus, in actuality multiple vertices can share the same normal vector.

Note: To turn light on or off after setting in the PolygonAttr command, first enable the set value with the Begin command and then issue the Normal command.

Where to issue the Normal command

The Normal command is usually issued between the Begin and End commands, but it can also be issued before the Begin command.

Command string Example 1	Begin→Normal→Vertex→Vertex→Vertex→End
Command string Example 2	Normal→Begin→Vertex→Vertex→Vertex→End→Begin→Vertex→Vertex→Vertex→End

VTX_16: Set the Vertex Coordinates

Name: VTX_16 Address: 0x0400048C Attribute: W Command Code: 0x23

31	30–28	27–16	15	14–12	11–0
SY	INT_Y	DECIMAL_Y	SX	INT_X	DECIMAL_X
Y Coordinate			X Coordinate		

Signed fixed-point number (sign + 3-bit integer + 12-bit fractional part)

31–16	15	14–12	11–0
	SZ	INT_Z	DECIMAL_Z
	Z Coordinate		

TWL-06-0017-001-D
Released: February 16, 2009

VTX_10: Set the Vertex Coordinates

Name: VTX_10 Address: 0x04000490 Attribute: W Command Code: 0x24

31		29	28 26	25 24 23 20	19	18 16	15 10	9	8 7	5 0
		SZ	INT_Z	DECIMAL_Z	SY	INT_Y	DECIMAL_Y	SX	INT_X	DECIMAL_X
		Z Coordinate			Y Coordinate			X Coordinate		

Signed fixed-point number (sign + 3-bit integer + 6-bit fractional part)

VertexXY: Set the Vertex XY Coordinates (for Z Coordinate, Use the Last-Set Data)

Name: VTX_XY Address: 0x04000494 Attribute: W Command Code: 0x25

31	30 28	27 24 23 16	15	14 12	11 8 7 0
SY	INT_Y	DECIMAL_Y	SX	INT_X	DECIMAL_X
Y Coordinate			X Coordinate		

Signed fixed-point number (sign + 3-bit integer + 12-bit fractional part)

VertexXZ: Set the Vertex XZ Coordinates (for Y Coordinate, Use the Last-Set Data)

Name: VTX_XZ Address: 0x04000498 Attribute: W Command Code: 0x26

31	30 28	27 24 23 16	15	14 12	11 8 7 0
SZ	INT_Z	DECIMAL_Z	SX	INT_X	DECIMAL_X
Z Coordinate			X Coordinate		

Signed fixed-point number (sign + 3-bit integer + 12-bit fractional part)

VertexYZ: Set the Vertex YZ Coordinates (for X Coordinate, Use the Last-Set Data)

Name: VTX_YZ Address: 0x0400049C Attribute: W Command Code: 0x27

31	30 28	27 24 23 16	15	14 12	11 8 7 0
SZ	INT_Z	DECIMAL_Z	SY	INT_Y	DECIMAL_Y
Z Coordinate			Y Coordinate		

Signed fixed-point number (sign + 3-bit integer + 12-bit fractional part)

VertexDiff: Set the Difference Value of the Last-Set Data for Vertex Coordinates

Name: VTX_DIFF Address: 0x040004A0 Attribute: W Command Code: 0x28

31	30	29	28–20	19	18–10	9	8–0
		SZ	DECIMAL_Z	SY	DECIMAL_Y	SX	DECIMAL_X
		Z Coordinate		Y Coordinate		X Coordinate	

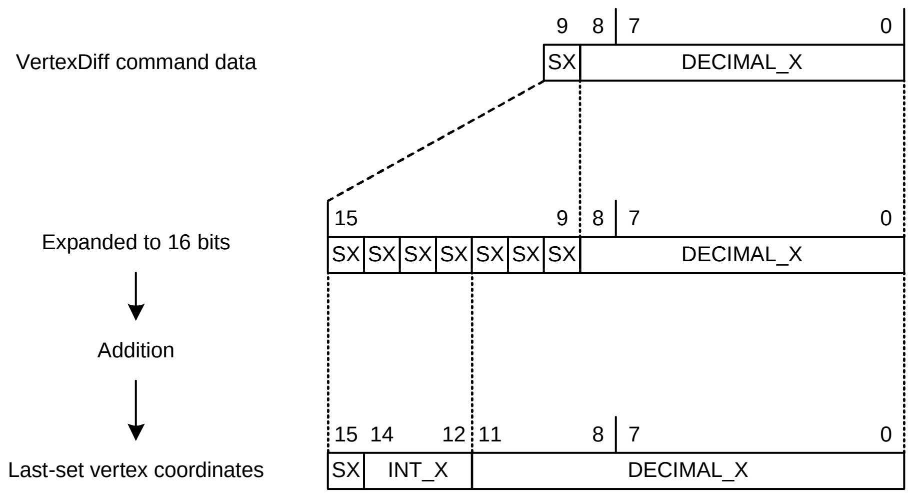
Signed fixed-point number (sign + 9-bit fractional part)

The value after adding to the last-set vertex value is stored as the 16-bit vertex coordinates.

The VertexDiff command data is sign-extended to 16 bits and added to the prior-set vertex coordinate. The vertex coordinates (the Vertex-group command's data) are 16 bits in size (sign + 3-bit integer + 12-bit fractional part), so the data of the VertexDiff command corresponds to the 4th to 12th places of the fractional part of the vertex coordinates (see Figure 7-22.)

Note: Use caution because an overflow can occur during the addition process.

Figure 7-22: The Process for Adding the X Coordinate



Items common to all Vertex commands

When Vertex commands are issued, the vertex data that have been transformed into BG screen coordinates are stored in Vertex RAM. Further, polygon data is stored to Polygon List RAM when the data for the number of vertices comprising the polygon are processed.

Note: Be sure to issue Vertex commands between the Begin command and the End command and that there are not too many or too few vertices in the specified primitives.

Cautions for polygons during clipping

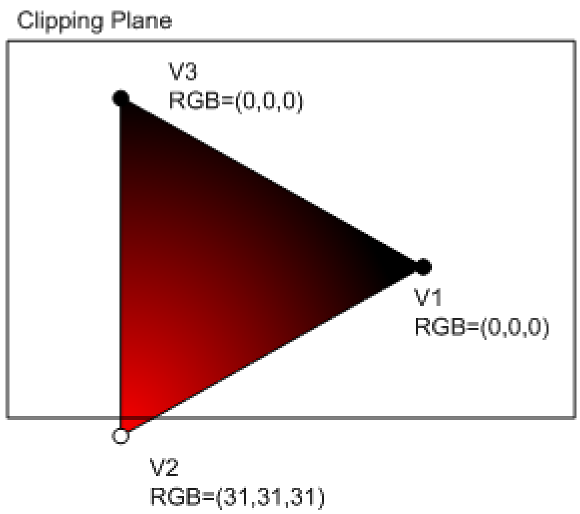
When a polygon is clipped, the G value, B value, or both on the clipping plane polygon will occasionally change to 0, causing color distortion. Figure 7-23 represents the state in which this has occurred (the G value and the B value have both changed to 0). When calculating the color of the vertex newly created through clipping, the color value may become a value higher than 31. When this happens, the last 5 bits of the value truncated to 32 will become the final color, incorrectly making it 0. This does not occur for the R value because the calculation accuracy is higher than the other two. This results in a reddish display.

Note: If the circuit revisions for the geometry circuit are enabled in TWL mode, this phenomenon will not occur (configure this with the SCFG_EXT register).

This issue may be avoided by using the following methods:

- Set the polygon scaling small to assure high geometry calculation accuracy.
- Shorten the vertex interval of the polygons or pull the vertex away from the clipping plane to reduce the effect of the errors stemming from low calculation accuracy.
- If the vertex color is directly configured through a modeling software, set the vertex color to be (R, G, B) = (31, 30, 30) or smaller.
- If the vertex color is not directly configured, adjust the material or light color so that the calculated vertex will be (R, G, B) = (31, 30, 30) or smaller.

Figure 7-23: Polygon Clipping Color Distortion



7.2.14 Texture Mapping

TexCoord: Set Texture Coordinates

Name: TEXCOORD Address: 0x04000488 Attribute: W Command Code: 0x22

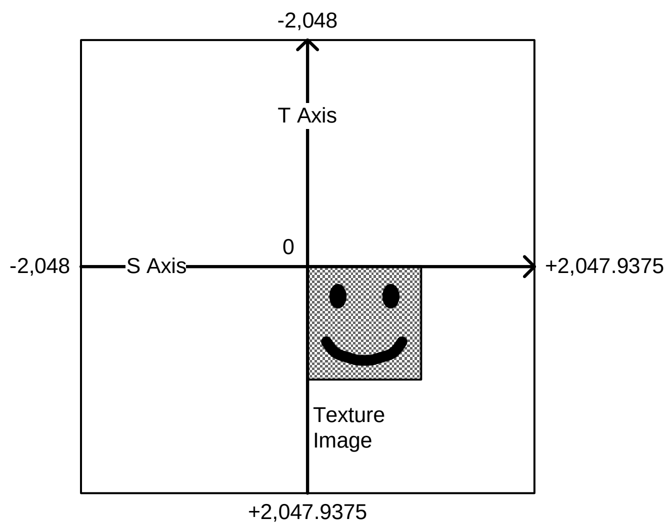
31	30 … 24 23	16	15	14 … 8 7	0
ST	INTEGER_T	DECIMAL_T	SS	INTEGER_S	DECIMAL_S
T Coordinate			S Coordinate		

Signed fixed-point number (sign + 11-bit integer + 4-bit fractional part)

- TEX_T, TEX_S[d31–d16], [d15–d00]: Texture coordinates

  As Figure 7-24 shows, the texture coordinates set the coordinates in texture image space, treating the texel size as 1.0 (4-bit fractional part).

Figure 7-24: Texture Image Space (for an Image of 1,024x1,024 Texels)



The texture coordinates remain valid until the next TexCoord command resets the current texture coordinates.

Because of this, the same texture coordinates can be shared by multiple vertices.

<u>Where to issue the TexCoord command</u>

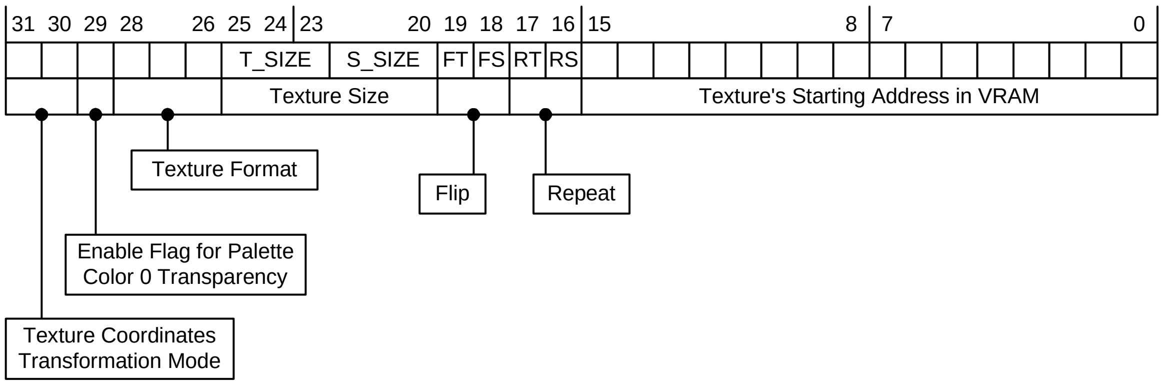
The TexCoord command is normally issued between the Begin command and the End command, but it can also be issued before the Begin command.

Command String Example 1	Begin→TexCoord→Vertex→Vertex→Vertex→End
Command String Example 2	TexCoord→Begin→Vertex→Vertex→Vertex→End→Begin→Vertex→Vertex→Vertex→End

When texture mapping, the Geometry Engine works faster if you issue commands in the following order: TexCoord→Normal→Vertex.

TexImageParam: Setting the Texture Parameters

Name: TEXIMAGE_PARAM Address: 0x040004A8 Attribute: W Command Code: 0x2A



- TGEN[d31–d30] : Texture coordinate transformation mode

00	Do not transform texture coordinates
01	TexCoord source
10	Normal source
11	Vertex source

- TR[d29] : Enable flag for the palette's color0 transparency

  When using transparent texels with 4-, 16-, and 256-color palette textures, set this bit to 1 so the palette's color0 can be referenced for the transparent color.

0	Enable the palette's color0 setting
1	Make appear transparent, regardless of the palette color0 setting value

- TEXFMT[d28–d26] : Texture format

0	No texture
1	A3I5 translucent texture
2	4-color palette texture
3	16-color palette texture
4	256-color palette texture
5	4x4 texel compressed texture
6	A5I3 translucent texture
7	Direct texture

- [d25–d20] : Texture size
  - T_SIZE, S_SIZE

    Selects texture size of 8 x 8 to 1,024 x 1,024.

0	8 texels
1	16 texels
2	32 texels
3	64 texels
4	128 texels
5	256 texels
6	512 texels
7	1,024 texels

- [d19–d18] : Flip

  Specifies whether to flip the texture image up/down and/or left/right for mapping when the texture coordinates pertain to a region beyond the texture size. (See Figure 7-25 and Figure 7-26.) The flip setting is valid only when Repeat has been specified.
  - FT[d19] : Flip in direction of T coordinates

0	Do not flip
1	Flip

  - FS[d18] : Flip in direction of S coordinates

0	Do not flip
1	Flip

- [d17–d16] : Repeat

  Specifies whether to repeatedly map the texture image when the texture coordinates pertain to a region beyond the texture size.
  - RT[d17] : Repeat in the direction of the T coordinates

0	Do not repeat
1	Repeat

  - RS[d16] : Repeat in the direction of the S coordinates

0	Do not repeat
1	Repeat

- TEX_ADDR[d15–d00] : Texture's starting address in VRAM

  The system references the 3-bit left-shift of the texture's starting address in VRAM.

Where to issue the TexImageParam command

The `TexImageParam` command is normally issued before the `Begin` command, but it can also be issued between the `Begin` and `End` commands. By issuing it during the Begin-End interval you can set different texture parameters for every polygon inside the `Begin-End` command string.

Note: A problem causes the previous polygon texture attributes to be overwritten with the parameters passed by the `TexImageParam` command, according to the process status of the geometry engine.

Note: If the circuit revisions for the geometry circuit are enabled in TWL mode, this phenomenon will not occur (configure this with the SCFG_EXT register).

The following examples show the positions in which to issue the `TexImageParam` command:

- Outside the Begin-End interval:

```
TexImageParam;
Begin;
        TexCoord;
        Vertex;
        ...
End;
```

- Within the Begin-End interval with quadrilateral polygons:

  1. When the command structure for the polygon that has changed texture after the second polygon changes from TexCoord to Vertex:

```
Begin;
        TexImageParam;
        First Polygon;
        TexImageParam;
        Normal;                 Send the Normal command as a dummy command
        Second Polygon;
        ...
End;
```

  2. When the command structure for the polygon that has changed texture after the second polygon changes from TexCoord to Normal to Vertex:

     The problem can be resolved effectively by sending 0xFF (undefined) as the dummy command, in place of the `Normal` command sent in the first example. However, when the `TexPlttBase` command is send with the `TexImageParam` command, this problem does not occur, and there is no need to send a dummy command.

- Within the Begin-End interval with triangle polygons:

  1. When the command structure for the polygon that has changed texture after the second polygon changes from TexCoord to Vertex:

     Same as example 1 for quadrilateral polygons.

  2. When the command structure for the polygon that has changed texture after the second polygon changes from TexCoord to Normal to Vertex:

     No problems occur.

- Within the Begin-End interval with triangle and quadrilateral polygons:

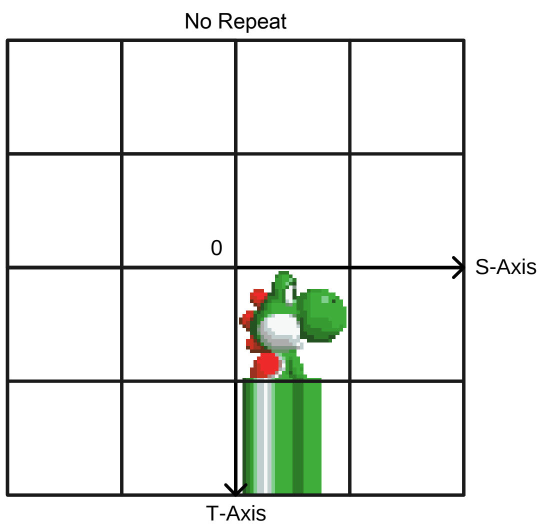
  In this case, always use `End` once when changing textures. In other words, limit issuing the `TexImageParam` command to a position before the Vertex command is issued, as shown below.

```
Begin;
        TexImageParam;
        Vertex;
        ...
End;
```

- Texture Flip and Repeat Settings (For a Texture Image of 1,024 x 1,024 Texels)

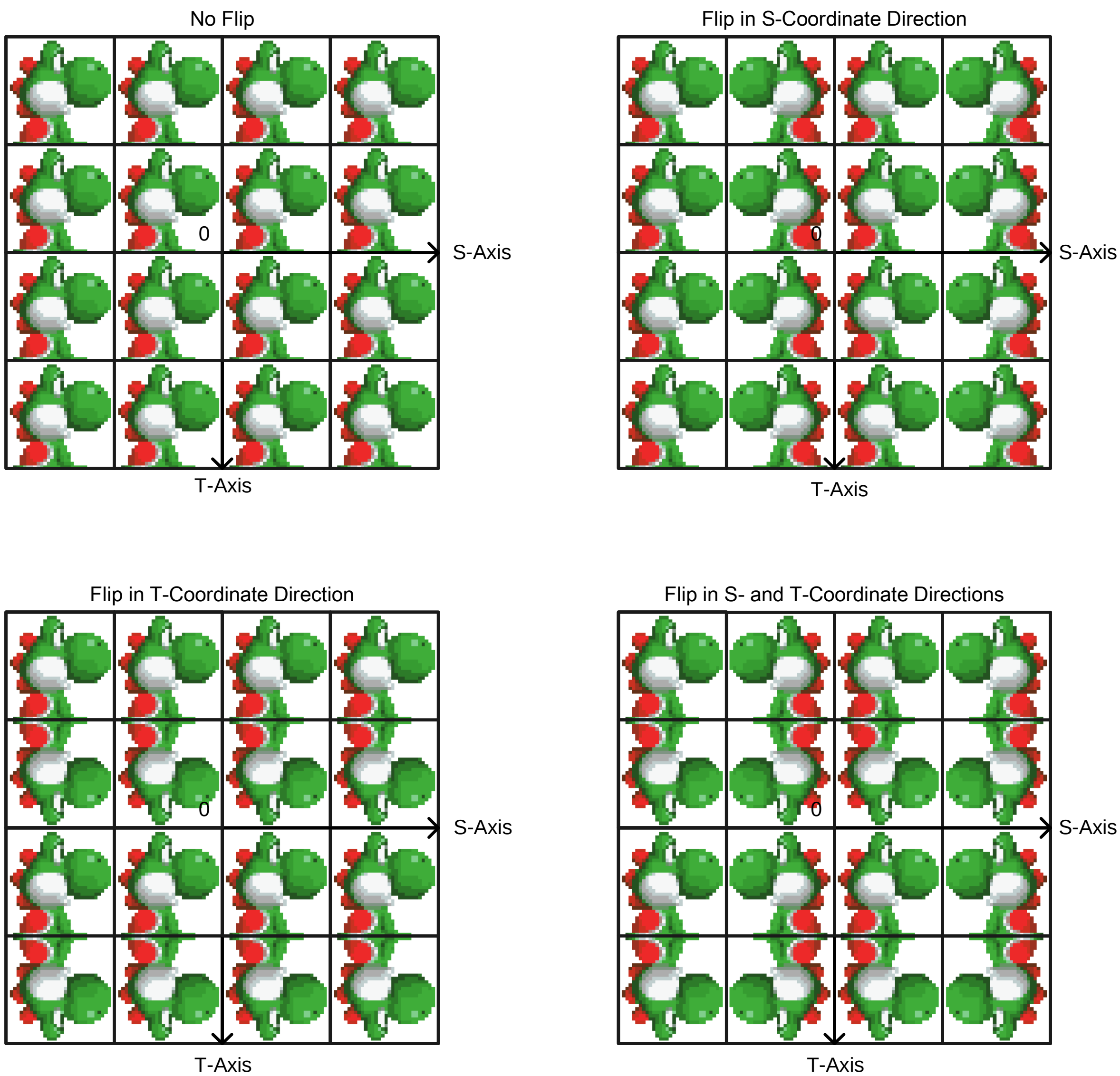
1. When when there is no repeat

Figure 7-25: Texture Image Space (No Repeats)



2. Flip settings when there is a repeat

Figure 7-26: Texture Image Space (with Repeats)



TexPlttBase: Set Texture Palette's Base Address

Name: TEXPLTT_BASE Address: 0x040004AC Attribute: W Command Code: 0x2B

31–13	12–0
	PLTT_BASE
	Palette's base address

- PLTT_BASE[d12–d00] : Specifies the palette's base address

  The system references a 2 to 4 bit left shift of the palette's base address.

  The shift volume varies, depending on the texture format (see "Table 7-4 : Number of Geometry Command Run Cycles & Timing Related to Command Issue (in Command Code Order)" on page 199).

Table 7-5: PLTT_BASE Values and Shift Volumes

PLTT_BASE Value	4-color Palette Texture	16-color Palette Texture	256-color Palette Texture	4x4 texels Compressed Texture	A3I5 Texture	A5I3 Texture
0x0000	0x00000	0x00000	0x00000	0x00000	0x00000	0x00000
0x0001	0x00008	0x00010	0x00010	0x00010	0x00010	0x00010
0x0002	0x00010	0x00020	0x00020	0x00020	0x00020	0x00020
0x17FE	0x0BFF0	0x17FE0	0x17FE0	0x17FE0	0x17FE0	0x17FE0
0x17FF	0x0BFF8	0x17FF0	0x17FF0	0x17FF0	0x17FF0	0x17FF0
0x1800	0x0C000	Setting prohibited	Setting prohibited	Setting prohibited	Setting prohibited	Setting prohibited
0x1FFE	0x0FFF0	Setting prohibited	Setting prohibited	Setting prohibited	Setting prohibited	Setting prohibited
0x1FFF	0x0FFF8	Setting prohibited	Setting prohibited	Setting prohibited	Setting prohibited	Setting prohibited

Where to issue the TexPlttBase command

The `TexPlttBase` command is normally issued before the `Begin` command, but it can also be issued between the `Begin` and `End` commands. By issuing it between these two commands you can set a different palette base address for every polygon between the `Begin` and `End` commands.

Command String Example 1	TexPlttBase→Begin→TexCoord→Vertex→TexCoord→Vertex→TexCoord→Vertex→End
Command String Example 2	Begin→TexPlttBase→TexCoord→Vertex→TexCoord→Vertex→TexCoord→Vertex→TexPlttBase→TexCoord→Vertex→TexCoord→Vertex→TexCoord→Vertex→End

7.2.14.1 Texture Coordinate Transformations

The texture coordinate transformation mode can be switched using the `TexImageParam` command.

In the three modes described in this section, the values given by the pertinent command are used as the input coordinates for the calculations.

The values after the coordinate transformation are not meant to be used, so make sure you set the texture matrix appropriately in advance.

<u>TexCoord source</u>

The texture coordinate transformation is performed using the values set by the `TexCoord` command as the input coordinates.

The coordinate transformation is executed when the `TexCoord` command is issued.

You can produce a simple texture scroll by setting a translation matrix or a rotation matrix for the texture matrix.

Operation Expressed in Matrix Form

$$\begin{bmatrix} S' & T' & R' & Q' \end{bmatrix} = \begin{bmatrix} S & T & \frac{1}{16} & \frac{1}{16} \end{bmatrix} \begin{bmatrix} m[0] & m[1] & m[2] & m[3] \\ m[4] & m[5] & m[6] & m[7] \\ m[8] & m[9] & m[10] & m[11] \\ m[12] & m[13] & m[14] & m[15] \end{bmatrix}$$

Specific Expression Taking Decimal-Point Position into Account

$$S' = \{m[0] \times (S « 12) + m[4] \times (T « 12) + m[8] \times (1 « 12) + m[12] \times (1 « 12)\} » 24$$
$$T' = \{m[1] \times (S « 12) + m[5] \times (T « 12) + m[9] \times (1 « 12) + m[13] + (1 « 12)\} » 24$$

<u>Normal source</u>

The texture coordinate transformation is performed using the values set by the `Normal` command as the input coordinates.

The coordinate transformation is executed when the `Normal` command is issued.

The S and T values set by the immediately-prior `TexCoord` command are used as the translation components of the texture coordinates.

You can produce spherical reflection mapping by setting in the texture matrix the result of reading the current directional vector matrix and multiplying by the scaling matrix that expands the directional vector space (-1.0 to 1.0) to 1/2 the texture size. When doing this, use the `TexCoord` command to translate the origin of the texture coordinate to the center of the spherical texture.

Operation Expressed in Matrix Form

$$\begin{bmatrix} S' & T' & R' & Q' \end{bmatrix} = \begin{bmatrix} Nu & Nv & Nw & 1 \end{bmatrix} \begin{bmatrix} m[0] & m[1] & m[2] & m[3] \\ m[4] & m[5] & m[6] & m[7] \\ m[8] & m[9] & m[10] & m[11] \\ S & T & m[14] & m[15] \end{bmatrix}$$

Specific Expression Taking Decimal-Point Position into Account

$$S' = \{m[0] \times (Nx \ll 3) + m[4] \times (Ny \ll 3) + m[8] \times (Nz \ll 3) + (S \ll 12) \times (1 \ll 12)\} \gg 24$$

$$T' = \{m[1] \times (Nx \ll 3) + m[5] \times (Ny \ll 3) + m[9] \times (Nz \ll 3) + (T \ll 12) \times (1 \ll 12)\} \gg 24$$

Vertex source

The texture coordinate transformation is performed using the values set by a Vertex-group command as the input coordinates. The coordinate transformation runs when a Vertex-group command is issued.

The S and T values set by the `TexCoord` command issued immediately prior are used as the translation components of the texture coordinates.

You can produce texture scrolls dependent on the View coordinates by reading the current position coordinate matrix and setting it to the texture matrix.

Operation Expressed in Matrix Form

$$\begin{bmatrix} S' & T' & R' & Q' \end{bmatrix} = \begin{bmatrix} X & Y & Z & 1 \end{bmatrix} \begin{bmatrix} m[0] & m[1] & m[2] & m[3] \\ m[4] & m[5] & m[6] & m[7] \\ m[8] & m[9] & m[10] & m[11] \\ S & T & m[14] & m[15] \end{bmatrix}$$

Specific Expression Taking Decimal-Point Position into Account

$$S' = \{m[0] \times X + m[4] \times Y + m[8] \times Z + (S \ll 12) \times (1 \ll 12)\} \gg 24$$

$$T' = \{m[1] \times X + m[5] \times Y + m[9] \times Z + (T \ll 12) \times (1 \ll 12)\} \gg 24$$

- **Decimal point positions between parameters used by texture-coordinate transformation expressions**

  The parameter formats used in the expressions for calculating texture coordinate transformations are shown in the following table.

  Therefore, to unify texture coordinate transformation calculations to 12-bit fractional parts, the Normal coordinates could be left-shifted by 3 bits, and the texture coordinates could be left-shifted by 8 bits before the calculation is applied.

  However, as can be seen in “Specific expression taking decimal-point position into account,” texture coordinates are left-shifted by 12 bits before the expression is applied, and the results that are right-shifted by 24 bits are taken as the new texture coordinates.

  From this, it is natural to assume that the texture coordinate transformation is (Sign + 15-bit integer + 0-bit fractional part). That is, texture coordinate transformation calculations use units of 1/16 texel, rather than units of 1 texel.

Parameter Name	Parameter	Format
Texture Matrix	**m[num] (num = 0-15)**	Sign + 19-bit integer + 12-bit fractional part
Normal Coordinate	**Nx, Ny, Nz**	Sign + 9-bit fractional part
Vertex Coordinate	**X, Y, Z**	Sign + 3-bit integer + 12-bit fractional part
Texture Coordinate	**S, T**	Sign + 11-bit integer + 4-bit fractional part

TWL-06-0017-001-D
Released: February 16, 2009

- **Technical tip**

  There may be times when you are using polygons to represent 2D graphics, and you want the texels to have 1:1 correspondence with the pixels on the LCD. Because texture sampling proceeds from the upper left texel, the texture may be off by 1 texel due to such factors as polygon rotation. To prevent this from happening, use a texture-coordinate transformation to adjust the position from the position at which sampling starts. For details, see "7.3.5.1.1 Texture Image Sampling" on page 269.

- **Polygon processing cycle count**

  When there are two lights or fewer, the execution cycle of the geometry engine will not vary based on the availability of the Normal command. When there are three or more lights, the execution cycle will be faster without the Normal command. Also, regardless of the number of lights, the transfer time (bus usage time) to the command FIFO will be shorter if the Normal command is not executed.

  The following table values result from the fact that when the number of lights is small, the coordinate conversion will become the bottleneck even if the calculation cycle of the vertex color is short. Therefore the polygon operation will be fixed to the vertex coordinate conversion time. But when the number of lights increase, the vertex color calculation cycle will exceed this time.

Command Structure	Light Count for Triangular Polygons					Light Count for Quadrilateral Polygons				
	OFF	1ch	2ch	3ch	4ch	OFF	1ch	2ch	3ch	4ch
TexCoord -> Normal -> Vertex	28	28	28	30	33	37	37	37	40	44
Normal -> Vertex	28	28	28	28	30	37	37	37	37	40
TexCoord -> Vertex	28	28	28	28	28	37	37	37	37	37
Vertex	28	28	28	28	28	37	37	37	37	37

7.2.15 Tests

When the Geometry Engine executes a test-related command (such as the status flag and the resulting register value), it updates the test command result.

BoxTest: Test if Cuboid Sits Inside View Volume

Name: BOX_TEST   Address: 0x040005C0   Attribute: W   Command Code: 0x70

31	30–28	27–16	15	14–12	11–0
SY	INT_Y	DECIMAL_Y	SX	INT_X	DECIMAL_X
Y Coordinate			X Coordinate		

Signed fixed-point number (sign + 3-bit integer + 12-bit fractional part)

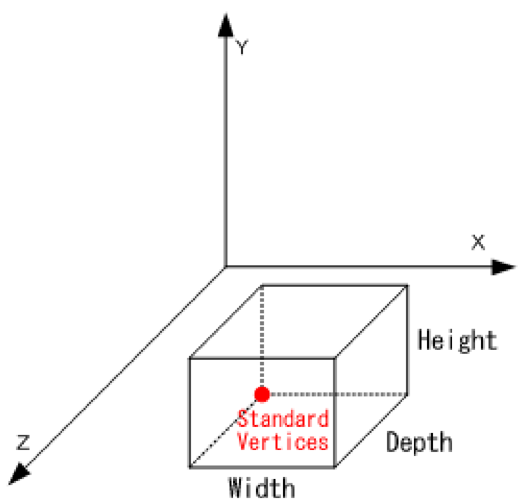
31	30–28	27–16	15	14–12	11–0
SW	INT_W	DECIMAL_W	SZ	INT_Z	DECIMAL_Z
Width			Z Coordinate		

31	30–28	27–16	15	14–12	11–0
SD	INT_D	DECIMAL_D	SH	INT_H	DECIMAL_H
Depth			Height		

Specify the box's standard vertices of the box shown in Figure 7-27 for the coordinate values.

The result of the Box test is stored in the Geometry Engine Status register (GXSTAT).

Figure 7-27: Box to Be Tested



- **Box Test Contents**

  Determines whether any of the six faces of the box are not completely within the view volume.

  For this reason, if the view volume is completely contained within the box, it is considered out of view.

  Note: If the circuit revisions for the geometry circuit are enabled in TWL mode, this phenomenon will not occur (configure this with the SCFG_EXT register).

- **Conditions Required to Properly Run a Box Test**

  Conduct the box test with both polygon attribute flags set to 1. If either of the flags is set to 0, the test results may not be correct.

  Keep in mind that overflow may occur when width/height/depth is added to the reference vertices. (The result of the addition must be greater than or equal to – 8.0 and less than 8.0.)

  1. Set both of the polygon attribute flags to 1:
     - Far plane-intersecting polygon display specification
     - 1-dot polygon rendering specification
  2. `Begin` command
  3. `End` command
  4. `BoxTest` command

PositionTest: Set the Position Coordinates for the Tests

Name: POS_TEST Address: 0x040005C4 Attribute: W Command Code: 0x71

31	30–28	27–16	15	14–12	11–0
SY	INT_Y	DECIMAL_Y	SX	INT_X	DECIMAL_X
Y Coordinate			X Coordinate		

Signed fixed-point number (sign + 3-bit integer + 12-bit fractional part)

31–16	15	14–12	11–0
	SZ	INT_Z	DECIMAL_Z
	Z Coordinate		

Coordinate transformation is performed on the position coordinates for the test by the current clip coordinate matrix.

The results of the Position test (the clip coordinates) are stored in the PositionResult register.

VectorTest: Set the Directional Vector for the Tests

Name: VEC_TEST Address: 0x040005C8 Attribute: W Command Code: 0x72

31–30	29	28–20	19	18–10	9	8–0
	SZ	DECIMAL_Z	SY	DECIMAL_Y	SX	DECIMAL_X
	Z Component		Y Component		X Component	

Signed fixed-point number (sign + 9-bit fractional part)

Coordinate transformation is carried out on the directional vector for the test by the current directional vector matrix.

The result of the Vector test (the directional vector in the View coordinate space) is stored in the VectorResult register.

PositionResult: Read the PositionTest Computational Results

Name	Address	Attribute	Initial Value
POS_RESULT_x (x=X,Y,Z,W)	0x04000620, 0x04000624, 0x04000628, 0x040062C	R	0x00000000

31	30–12	11–0
Sx	INTEGER_x	DECIMAL_x
Sign	Integer Part	Decimal Part

Signed fixed-point number (sign + 19-bit integer + 12-bit fractional part)

The clip coordinates values (x, y, z, w) are stored in these registers.

VectorResult: Read the VectorTest Computational Results

Name	Address	Attribute	Initial Value
VEC_RESULT_x (x=X,Y,Z)	0x04000630, 0x04000632, 0x04000634	R	0x0000

15	12–8	7–0
Sx	INTEGER_x	DECIMAL_x
Sign	Integer Part	Decimal Part

Signed fixed-point number (sign + 3-bit integer + 12-bit fractional part)

The directional vector values (x, y, z) of the view coordinate space are stored in these registers.

The read-out computation results are within ±1, so the integer is a sign extension.

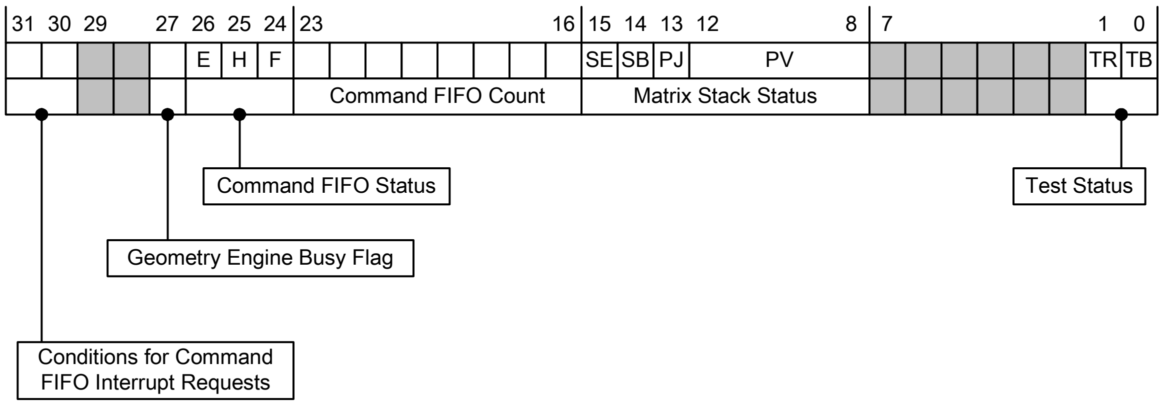
7.2.16 Status

You can check the status of Polygon List RAM and Vertex RAM using the previously mentioned DISP3DCNT register.

To check the status of command FIFO and matrix stacks, etc., see the GXSTAT register diagram below.

GXSTAT: Geometry Engine Status Register

Name: GXSTAT Address: 0x04000600 Attribute: R/W Initial Value: 0x00000000



- FI[d31–d30] : Conditions for Command FIFO interrupt requests

00	Disable Command FIFO interrupt requests
01	Make interrupt request when Command FIFO is less than half full
10	Make interrupt request when Command FIFO is empty
11	Setting prohibited

- B[d27] : Geometry Engine busy flag

0	Geometry Engine is stopped
1	Geometry Engine is running

If commands or parameters have not been sent, the busy flag goes to the command wait / parameter wait status with the busy flag set to 0.

When commands or parameters resume, geometry processing resumes.

When a SwapBuffers command is issued, and no subsequent commands are issued, the geometry engine busy flag is set to 0 at the completion of a `SwapBuffers` process, which begins with a V-Blank. (It is 1 until then.)

This timing occurs 400 cycles (calculated at 33 MHz) after the V-Blank.

If, after the `SwapBuffers` command is issued, a `next` command is also issued, the `next` command is executed after the completion of the `SwapBuffers` process, which commences after the V-Blank.

- [d26–d24] : Command FIFO status
  - E[d26] : Command FIFO empty flag

0	FIFO is not empty
1	FIFO is empty

  - H[d25] : Command FIFO under-half flag

0	FIFO is at least half full
1	FIFO is less than half full

  - F[d24] : Command FIFO full flag

0	FIFO is not full
1	FIFO is full

- [d23–d16] : Command FIFO count value

  Can reference the number of commands/amount of data currently stored in Command FIFO.

- [d15–d08] : Matrix stack status
  - SE[d15] : Stack error flag

    The flag is set to 1 when an overflow or underflow of the matrix stack occurs.

    It can be cleared by writing 1.

0	No stack overflow or underflow
1	Stack overflow or underflow

When referencing the status error flag of matrix stack status, confirm that the `PushMatrix` and `PopMatrix` commands that have been issued have completed by first referencing the matrix stack busy flag.

  - SB[d14] : Matrix stack busy flag
  - When referencing PJ and PV matrix stack levels, check this flag to confirm that execution of the issued `PushMatrix` or `PopMatrix` command has completed.

0	There is no unexecuted `PushMatrix` or `PopMatrix` command.
1	The `PushMatrix` or `PopMatrix` command has been issued, and the execution is not yet completed.

  - PJ[d13] : Projection matrix stack level

    Can reference the current stack level (0–1)

  - PV[d12–d08] : Position and Vector matrix stack level

    Can reference the current stack level (0–31)

- [d01–d00] : Test status flag
  - TR[d01] : Box test result

0	All the faces that constitute the box are outside the view volume.
1	Of the six faces that constitute the box, part of one of the faces, or all are inside the view volume.

  - TB[d00] : Test busy flag

    Can reference the ready/busy status of each test (BoxTest, PositionTest, VectorTest).

0	Ready
1	Busy

LISTRAM_COUNT: Polygon List RAM Count Register

Name: LISTRAM_COUNT Address: 0x04000604 Attribute: R Initial Value: 0x0000

15				11			8	7							0
				Polygon-List RAM Counter											

- [d11–d00] : Polygon List RAM counter (Maximum valid value is 0x800)

  You can reference the number of opaque polygons + translucent polygons that are currently stored in Polygon List RAM.

  Polygon List RAM has a capacity of 2048 polygons, so the maximum valid value is 0x800.

  The polygon list RAM counter is cleared 10 system clock cycles (33.5MHz) after the V-Blank that comes immediately after the `SwapBuffers` command is issued.

VTXRAM_COUNT: Vertex RAM Count Register

Name: VTXRAM_COUNT Address: 0x04000606 Attribute: R Initial Value: 0x0000

15			12				8	7							0
			Vertex RAM Counter												

- [d12–d00] : Vertex RAM counter (Maximum valid value is 0x1800)

  You can reference the number of vertices that are currently stored in Vertex RAM.

  Vertex RAM has a capacity of 6144 vertices, so the maximum valid value is 0x1800.

  The vertex RAM counter is cleared 10 system clock cycles (33.5MHz) after the V-Blank that comes immediately after the `SwapBuffers` command is issued.

7.2.16.1 Data Storage Capacity of Polygon List RAM and Vertex RAM

1. Polygon List RAM

The 52 KB of Polygon List RAM is divided into an ORDER area of 12 KB, followed by a POLYGON area of 40 KB. The number of polygons that can be stored in these two areas of Polygon List RAM does not change when polygons are connected, but because connected polygons share vertices, the same number of polygons consume less memory if they are connected. This is an effective way to economize on Vertex RAM.

No matter how the polygons are drawn, the ORDER area of Polygon List RAM can store 2048 polygons. However, the storage capacity of the POLYGON area varies, depending on the conditions under which polygons are drawn. In this area, the polygon data comprises a header region of 12 bytes followed by a vertex index region of 8 or more bytes. Normally, each triangular polygon consumes 8 bytes of the vertex index region, and each quadrilateral polygon consumes 12 bytes. Accordingly, the maximum number of polygons that can be stored in the POLYGON area is calculated as follows:

(Connected) triangular polygons: 40 KB / (12 byte header + 8 bytes) = 2048 polygons

(Connected) quadrilateral polygons: 40 KB / (12 byte header + 12 bytes) = 1706 polygons

Note that 4 bytes in the vertex index region are consumed each time the number of vertices increases due to clipping. This means that the total number of polygons that can be stored in the POLYGON area decreases by one polygon for every five clippings performed on triangular polygons and for every six clippings performed on quadrilateral polygons.

2. Vertex RAM

The 72 KB of Vertex RAM is fully available to store vertex data. Each vertex consumes 12 bytes.

Table 7-6 shows the amount of vertex RAM consumed and the maximum number of polygons that can be stored for each primitive type.

Table 7-6: Vertex RAM Consumed and the Maximum Number of Polygons Stored per Primitive Type

Primitive Type	Vertex RAM Consumption	Maximum No. of Polygons that Can Be Stored
Triangle Polygon	3 vertices per polygon	2048
Quadrilateral Polygon	4 vertices per polygon	1536
Connected Triangle Polygons	First polygon: 3 vertices Later polygons: 1 vertex	6142 However, with 2050 vertices, the Polygon List RAM maximum of 2048 polygons is reached.
Connected Quadrilateral Polygons	First polygon: 4 vertices Later polygons: 2 vertices	3070 However, with 3414 vertices, the Polygon List RAM maximum of 1706 polygons is reached.

Because clipping also consumes Vertex RAM, connect polygons whenever possible as an effective method to avoid Vertex RAM overflow.

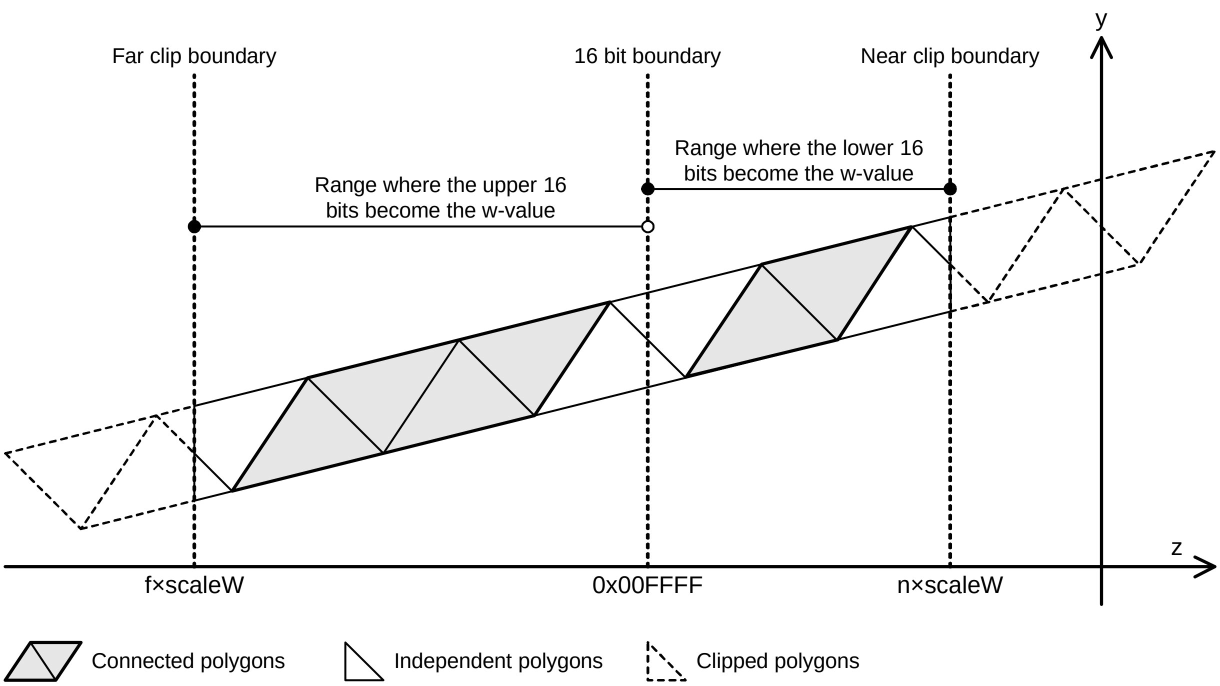
a. Why shared vertices are released during both Z-Buffering and W-Buffering

When clipping is performed on connected polygons, the shared vertices between neighboring polygons are released.

b. Why shared vertices are released only during Z-Buffering

During Z-buffering, shared vertices are released if the W values (twice the clip-coordinate W values) stored in Vertex RAM for neighboring connected polygons are such that the W value for even one vertex of one of the polygons exceeds 16 bits (15 bits in clip coordinates) while not exceeding 16 bits for any of the vertices of the other polygon. (See Figure 7-28.) This happens because the Z value and the W value are both stored in Vertex RAM during Z-buffering, so the 24-bit W value can only be stored with 16-bit precision. What is stored is either the upper 16 bits or the lower 16 bits of the 24-bit W value, with the result that differences arise between polygons. This situation does not arise during W buffering because there is no need to store the Z value to vertex RAM during W buffering. Therefore, the 24-bit W value of the clip coordinate will be stored as it is, and the release of shared vertices does not occur.

Figure 7-28: Release of Shared Vertices Among Connected Polygons (Clip Coordinate System)



z axis: 24-bit W value

To read about n, f, scaleW, 24-bit W values see "7.2.5 Depth Buffering" on page 187.

c. Reasons for released vertices in the polygon attribute settings for rendering 1-dot polygons

If "Do not render 1-dot polygons" is set in the polygon attributes and the polygon's W value exceeds the value in the 1-dot polygon display boundary depth value register, the vertices shared among the connected polygons will be released. In this case, whether or not a shared vertex will be released is decided for each polygon. Therefore, groups of polygons that do not exceed the value in the 1-dot polygon display boundary depth value register will share vertices unaltered. To avoid this, set "Display 1-dot polygons" in the polygon attributes.

Note: If the circuit revisions for the geometry circuit are enabled in TWL mode, the shared vertices will not be released (configure this with the SCFG_EXT register).

4. **Memory capacities and the storable number of polygons**

The table shows the relationship between the capacity of each memory region and the number of polygons that can be stored in each region.

Memory Region	Relational Expression for Storable No. of Polygons & Memory Capacity
The POLYGON Area of Polygon List RAM	(12+8) x (F3+MF3) + (12+12) x (F4+MF4) + 4 x CLIP <= 40KB
Vertex RAM	(12 x 3) x (F3+MP3+DMF3) + (12 x 1) x (MF3-MP3-DMF3) + (12 x 4) x (F4+MP4+DMF4) + (12 x 2) x (MF4-MP4-DMF4) + 12 x CLIP <= 72KB

F3: Number of triangular polygons

MF3: Number of connected triangular polygons

MP3: Number of primitives of connected triangular polygons

F4: Number of quadrilateral polygons

MF4: Number of connected quadrilateral polygons

MP4: Number of primitives of connected quadrilateral polygons

CLI : Number of times clipping occurs

DMF3: Number of times vertices released for connected triangular polygons

DMF4: Number of times vertices released for connected quadrilateral polygons

5. **Summary**

The actual number of polygons that are rendered is limited by the lesser of the number of polygons that can be stored in Polygon List RAM and in Vertex RAM.

When extensive use is made of polygon strips, Vertex RAM has plenty of space. Therefore, you can estimate, based on the capacity limitations of Polygon List RAM.

If you make extensive use of polygon strips, even in the unusual situation where every polygon is clipped once, you can be sure to obtain these numbers of polygons:

Triangular polygon strips: 40 KB / (12 + 8 + 4) = 1706 polygons

Quadrilateral polygon strips: 40 KB / (12 + 12 + 4) = 1462 polygons

7.2.17 Warnings Regarding Calculation Precision

Although you can specify a 32-bit space for the world coordinate system (sign + 19-bit integer + 12-bit fractional part) with TWL, objects on the edges of this 32-bit space may appear distorted and wrapped because the computational precision of the hardware is also 32 bits. To use the world coordinate system without problems, keep within a range of 29 bits (sign + 16-bit integer + 12-bit fractional part).

TWL also has a 24-bit view space (sign 1 bit + 11-bit integer + 12-bit fractional part). Objects can appear distorted and wrapped if you specify some other space as the view volume.

7.3 Rendering Engine

In TWL mode, it is possible to use the revised circuits for the following features that had problems on NITRO by means of the system configuration. However, in NITRO compatibility mode, it will be necessary to follow the notes to work around the problems.

- Shadow : Refer to "7.3.4.4 Shadow Polygons" on page 262.
- 4x4 Texel Compressed Textures : Refer to "7.3.5.2.1.4 4x4 Texel Compression Textures" on page 275.

7.3.1 Overview

Table 7-7 lists the rendering engine specifications.

Table 7-7: Rendering Engine Specification List

Operating frequency	33.514 MHz
Render data	Triangles and quadrilaterals
Rendering capacity	Maximum 120,000 polygons/sec (60FPS) Maximum 30 million pixels/sec (60FPS)
Shadow surface process	Switch between Z-value buffering and W-value buffering methods
Shading	Gouraud shading
Texture mapping	Perspective correction, modulation/decal Support for 4x4 texel compression Support for translucent textures Flip, Repeat Image sizes of 8x8 texels to 1024x1024 texels
Other capabilities (See Table 7-8)	Alpha blending, alpha test, anti-aliasing, edge marking, fog, toon shading, highlight shading, shadow, wireframe, Clear Image

Table 7-8 gives an overview of rendering engine features.

Table 7-8: Overview of Rendering Engine Features

Alpha Blending	Blends the color value stored in the color buffer with the input fragment's color value based on the alpha value of that fragment (the fragments after texture blending).
Alpha Test	Compares the fragment's alpha value with the reference value set in the register and draws only if the fragment's alpha value is larger than this reference value. (These fragments are the fragments after texture blending.)
Antialiasing	Blends the color of the polygon's boundaries with color values of the polygon behind it using the (5-bit width) factor computed based on the shift from the original display position.
Edge Marking	Marks the boundary edges of polygons with different polygon IDs (6-bit) using the polygon edge-specified color (8 colors). When anti-aliasing is enabled, the edge marking is followed by anti-aliasing.
Fog	Using the fog density table, the specified fog's color value is blended with the color buffer's color value. The fog density can be specified in 32 levels, and the value that is applied is the value that results from linear interpolation with the depth value of the target pixels. When 3D is displayed in front of a 2D screen, fog can be applied to the 2D screen as well by using the color buffer.
Toon Shading	Can present cartoon-like pictures by steepening the shininess calculation results.
Highlight Shading	Can present shininess beyond the texture color.
Shadow	Can easily put shadows on even bumpy surfaces by defining the shadow volume.
Wireframe	Can draw only the edges of polygons without drawing the surfaces.
Clear Image	Can apply Clear Images in VRAM as the initial values for the color buffer's, depth buffer's and attribute buffer's fog-enable flags.

Note: The rendering-related registers have a double-buffer structure, and the contents of each register are sent to the Rendering Engine at the start of the V-Blank period that begins right after the `SwapBuffers` command is issued. Therefore, data can be written to these registers even during the middle of a frame. The data is not reflected in the image drawn in the frame at the time of the change.

7.3.2 Rendering Methods

7.3.2.1 Line Buffer Rendering

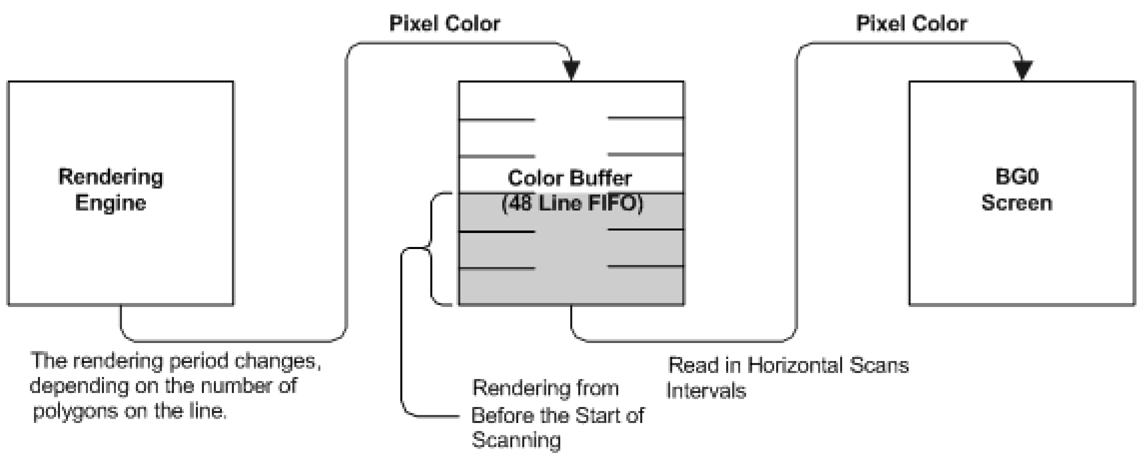
TWL employs a line-buffer method for rendering, rather than a frame-buffer method. Because of this, line overflow can occur when too many polygons are layered on a single line to be rendered during the horizontal period. To prevent line overflow from occurring easily, the color buffer operates FIFO, holding 48 lines, and the rendering begins from the middle of the V-Blank period, drawing and storing the line data before display (see Figure 7-29.)

By reading the RDLINES_COUNT register, you can check the minimum number of lines that remain in the color buffer while the frame that has been rendered is displaying. In other words, you cannot confirm whether or not lines have been dropped, but you can determine the risk of this happening.

To read about the RDLINES_COUNT register see "7.3.11 Status" on page 292.

This diagram shows only the FIFO operation. The Rendering Engine is actually reading from and writing to the color buffer.

Figure 7-29: Color Buffer's FIFO Operation



7.3.2.2 Buffers in the Rendering Engine

Table 7-9 shows the buffers in the Rendering Engine that store information about every pixel.

Also see "Figure 7-1 : 3D Graphics Hardware Block Diagram" on page 177.

Table 7-9: Buffers in the Rendering Engine

Stencil buffer	This buffer holds one line at 1 bit/pixel. It is used when rendering shadow polygons.
Attribute buffer	This buffer holds two lines at 23 bits/pixel. This buffer stores the polygon ID and fog enable flag for every pixel. Polygon IDs are managed separately for opaque polygons and translucent polygons.
Depth buffer	This buffer holds two lines at 24 bits/pixel. It is used for depth test and fog blending calculations.
Color buffer	This buffer holds 48 lines at 23 bits/pixel (R:G:B:A = 6:6:6:5).

7.3.2.3 Blank Periods

The Rendering Engine begins rendering when scanning starts for the LCD's 214th line, and it keeps rendering until the start of display of the 191st final line. Thus, the Rendering Engine's blank period is the 23 lines from line 191 to 213 (see Figure 7-30).

To safely rewrite data to the VRAM region (texture image and texture palette) referenced by the Rendering Engine, do so during these 23 lines.

Figure 7-30: Rendering Engine Blanking Periods

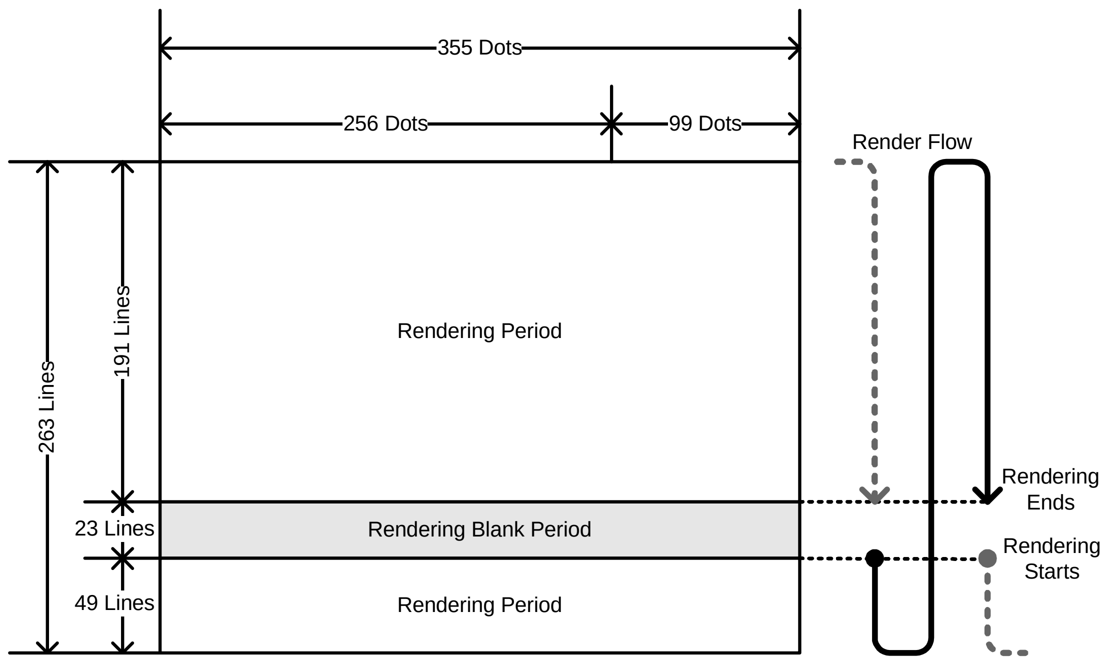


Table 7-10 shows the Rendering Engine timing specifications.

Table 7-10: Rendering Engine Timing Specifications

Item		Spec	Period
Rendering Period	Number of Horizontal Dots	256 dots	45.8316 µs
	Number of Vertical Lines	49+191 lines	15.2533 ms
Total Period	Number of Horizontal Dots	355 dots	63.5556 µs
	Number of Vertical Lines	263 lines	16.7151 ms
Blank Period	Number of blank Lines	23 lines	1.4618 ms
Scan Cycle	V Cycle	59.8261Hz	16.7151 ms

7.3.2.4 Number of Polygons that Can Be Drawn with One Line

The Rendering Engine can draw for a period of 355 dots with one line. (See "Table 7-10 : Rendering Engine Timing Specifications" on page 254).

Each dot involves 6 cycles, but since there is also an overhead of 4 cycles for each line, the number of cycles that can be used for rendering is (355 x 6) - 4 = 2126 cycles.

In TWL, the fill rate for even texture-mapped translucent polygons is 1 pixel per cycle, but there is an 8-cycle overhead every time rendering of a polygon starts.

Given these factors, the minimum guaranteed number of polygons that can be drawn on one line is:

(2126 cycles) / (8 cycles + number of horizontal pixels in polygon).

Excluding the overhead, the per-line fill rate is:

(2126 cycles) – (8 cycles x guaranteed number of polygons drawn with one line).

Using these formulas yields the results shown in Table 7-11.

Table 7-11: Maximum Polygons Rendered per Line and Fill Rate (Calculated Values)

Number of Horizontal Pixels in the Polygon	8	16	32	64	128	256
Number of Polygons Guaranteed to Be Drawn by One Line	132	88	53	29	15	8
Per-line Fill Rate	1070	1422	1702	1894	2006	2062

Because the line buffer is FIFO, the actual number of polygons drawn may be higher.

7.3.3 Initializing the Rendering Buffers

7.3.3.1 Initializing with the Clear Registers

ClearColorAttr: Clear Color Attribute Register

Name: CLEAR_COLOR Address: 0x04000350 Attribute: W Initial value: 0x00000000

31	29 … 24	23	20 … 16	15	14 … 10	9 8 7 … 5	4 … 0
				F	BLUE	GREEN	RED
	Clear Polygon ID		α value	Fog	Color		

- Clear Polygon ID [d29–d24] : Polygon ID initial value

  Sets the initial value of the opaque Polygon ID in the Attribute buffer. Whether edge marking is applied is determined by comparing the polygon ID and this Clear Polygon ID in the case of edges that can be clipped at the edge of the screen.

- α value[d20–d16] : Initial value of α

  Sets the initial value of the α value in the Color buffer.

  Normally, set this to 0 when compositing with 2D.

- F[d15] : Fog enable flag

  Sets the initial value of the fog enable flag in the Attribute buffer.

  When compositing with the 2D screen, you can use this to control whether or not to apply fog to the rear plane.

  This is effective when you want to clearly display a 2D background.

- Color[d14–d00] : Clear Color RGB values

  Sets the Color buffer's initial RGB values.

  The Color buffer in the Rendering Engine is (R:G:B = 6:6:6) bits, so the lower 1 bit is treated as 0 when the Clear Color value is 0, and as 1 when the value is non-zero.

ClearDepth: Clear Depth Register

Name: CLEAR_DEptH Address: 0x04000354 Attribute: W Initial value: 0x7FFF

15	14 … 8	7 … 0
	CLEARDEPTH	
	Clear Depth Value	

- CLEARDEptH[d14–d00] : Clear Depth value

  The Depth buffer in the Rendering Engine is 24 bits/pixel, so the Clear Depth value is used after shifting 9 bits to the left. The lower 9 bits are treated as 0, except when the Clear Depth value is 0x7FFF, in which case the lower 9 bits are treated as 1.

  To read how this differs from the depth value in the different depth buffering methods, see "7.2.5 Depth Buffering" on page 187.

7.3.3.2 Initializing with Clear Images

When the Clear Image Enable Flag is set in the 3D Display Control register (DISP3DCNT), the Clear Images stored in VRAM are used as the initial values of the Color buffer and Depth buffer and the Attribute buffer's fog enable flag. Even when this feature is used, the value of the CLEAR_COLOR register is still used for the Attribute buffer's polygon ID.

Use the RAM Bank Control register to assign the VRAM that stores Clear Images to the Clear Image buffer.

The format for each Clear Image is given below. "Figure 7-31 : VRAM Mapping of Clear Images (Texture Image Slots 2 and 3 Shared)" on page 258 shows Clear Image VRAM mapping.

Clear Color Image Format

15	14 … 10	9 … 5	4 … 0
A	BLUE	GREEN	RED
Clear Alpha Flag	Clear Color		

- [d15] : Clear α flag

  The actual clear α values are as follows:

0	0x00
1	0x1F

- [d14–d00] : Clear Color RGB values
- Sets the Color buffer's RGB initial values.
- The Color buffer in the Rendering Engine is (R:G:B = 6:6:6) bits, so the lower 1 bit is treated as 0 when the Clear Color value is 0, and as 1 when the value is non-zero.

Clear Depth Image Format

15	14 … 8	7 … 0
FOG		
Clear Fog	Clear Depth	

- FOG[d15]: Clear Fog

  Sets the initial value of the fog enable flag in the Attribute buffer.

  When compositing with the 2D screen, you can use this to control whether or not to apply fog to the rear plane.

  This is effective when you want to clearly display a 2D background.

•[d14–d00] : Clear Depth

To read how this differs from the depth value in the different depth buffering methods, see "7.2.5 Depth Buffering" on page 187

Figure 7-31: VRAM Mapping of Clear Images (Texture Image Slots 2 and 3 Shared)

	Dot 0	1	2	3		253	254	255
Line 0	0h	2h	4h	6h		1FAh	1FCh	1FEh
1	200h	202h	204h	206h		3FAh	3FCh	3FEh
2	400h	402h					5FCh	5FEh
3	600h	602h					7FCh	7FEh
4	800h							9FEh
251	1F600h							1F7FEh
252	1F800h	1F802h					1F9FCh	1F9FEh
253	1FA00h	1FA02h					1FBFCh	1FBFEh
254	1FC00h	1FC02h	1FC04h	1FC06h		1FDFAh	1FDFCh	1FDFEh
255	1FE00h	1FE02h	1FE04h	1FE06h		1FFFAh	1FFFCh	1FFFEh

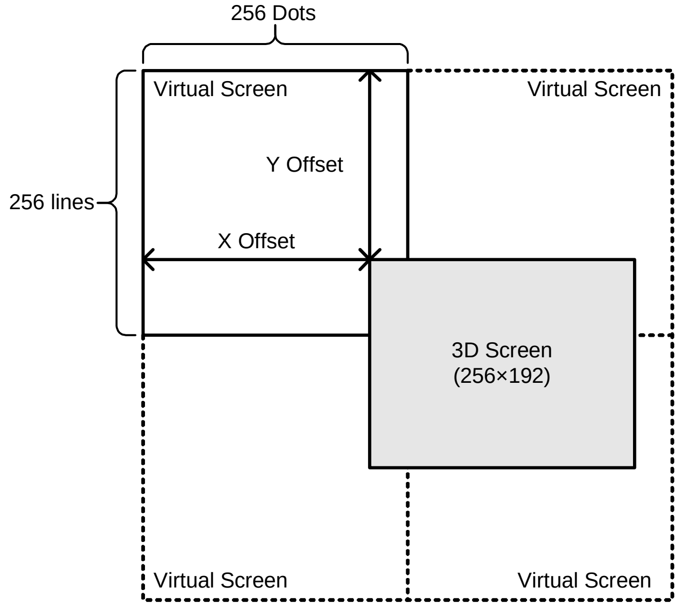
ClearImageOffset: Clear Image Offset Settings Register

Name: CLRIMAGE_OFFSET Address: 0x04000356 Attribute: W Initial value: 0x0000

15–8	7–0
Y Offset	X Offset

Can assign offsets to the Clear Images that are read when the rendering buffers are initialized. The image is wrapped and read if it exceeds the 256x256 data region on screen. Figure 7-32 shows the clear image offset.

Figure 7-32: Clear Image Offset



7.3.4 Rasterizing

Rasterization is the process of dividing the polygon surface into pixels and writing them to a buffer. During rasterization, the Rendering Engine interpolates the pixel colors inside the polygon, based on the vertex colors passed from the Geometry Engine.

The Rendering Engine stores the pixel colors in the Color buffer and stores each pixel's polygon ID and fog enable flag in the Attribute buffer. The rendering engine first finishes rendering the opaque polygon that are in polygon list RAM, and then renders the translucent polygons.

7.3.4.1 Opaque Polygons

An opaque polygon is a polygon with an α value of 31 ($\alpha = 31$).

- Polygon ID

  This is stored in the region for opaque polygon IDs in the Attribute buffer when the polygon is rendered.

- Fog enable flag

  When an opaque polygon is rendered, the fragment's fog enable flag overwrites the Attribute buffer's fog enable flag.

7.3.4.2 Translucent Polygons

A translucent polygon has either an α value between 1 and 30 ($1 \leq \alpha \leq 30$) or a translucent texture applied.

Therefore, this term also encompasses shadow polygons.

7.3.4.2.1 Polygons with $1 \leq \alpha \leq 30$

- Polygon ID

  This is stored in the region for translucent polygon IDs in the Attribute buffer when the polygon is rendered.

  When different translucent polygons overlap on the screen, a translucent polygon that uses the same polygon ID as another translucent polygon is not overwritten.

- Fog enable flag

  When a translucent polygon is rendered, the fragment's fog enable flag and the Attribute buffer's fog enable flag are combined with a logical AND operation, and the result is written to the Attribute buffer.

  This feature can be used to apply fog to everything except specific translucent polygons.

7.3.4.2.2 Translucent Texture-Mapped Polygons

Translucent texture-mapped polygons (A3I5 and A5I3 textures) are stored in the translucent polygon region of the polygon list RAM, even if all of the texels are opaque. Therefore, they are rendered after opaque polygons. Notice that processing of polygon IDs and fog enable flags differs, according to whether the pixel is opaque (α=31) or translucent ($1 \le \alpha \le 30$). Therefore, opaque pixels and translucent pixels may be mixed within a polygon.

- Polygon ID
  - Opaque pixels

    When rendering polygons, these are stored in the attribute buffer's opaque polygon ID area.

    Therefore, they are targeted for edge marking.
  - Translucent pixels

    When rendering polygons, they are stored in the attribute buffer's translucent polygon ID area.

    If different translucent polygons overlap on the screen, translucent polygons that have the same polygon ID are not overwritten.
- Fog enable flag
  - Opaque pixels

    Fragment fog enable flags are overwritten by the attribute buffer's fog enable flags.
  - Translucent pixels

    When a translucent polygon is drawn, the fragment's fog enable flag and the Attribute buffer's fog enable flag are combined with a logical AND operation, and the result is written to the Attribute buffer.

    This feature can be used to apply fog to everything except specific translucent polygons.

7.3.4.3 Wireframes

A *wireframe* is a polygon with an α *value* of 0 (α = 0). In this case, α does not have its original meaning of opacity level. Instead, only the outline of the polygon (wireframe) is rendered. If it is clipped, the *clipping boundary* (a new side created due to clipping) is also rendered as a wireframe. To render a wireframe as translucent, map a translucent texture.

Note: The characteristics of the circuit do not permit a wireframe to be drawn semi-transparently.

About the Polygon ID

When a wireframe is rendered, it is stored in the opaque polygon ID region of the attribute buffer. (Only the polygon ID of the wire is updated.)

7.3.4.4 Shadow Polygons

Shadow volume is defined as the space that is not illuminated by light because the light has been obstructed by an object. A shadow, then, can be thought of as something that is generated in the region where the shadow volume intersects with another object. The polygon used to express a shadow volume is called a *shadow polygon*.

You can create a shadow image that can be seen from the user's perspective by creating only a mask image of the Stencil buffer when rendering the inner side of the shadow volume, and then excluding the mask region when drawing outside the shadow volume.

Shadow polygons used for masks and rendering can be differentiated by their polygon IDs.

Polygon ID	Classification
0	Shadow polygons for masks
1–63	Shadow polygons for drawing

Figure 7-33 illustrates the concept of shadow polygons. "Figure 7-34 : When Drawing a Shadow Polygon for a Mask" on page 264 describes rendering shadow polygons for masking. "Figure 7-34 : When Drawing a Shadow Polygon for a Mask" on page 264 describes rendering shadow polygons for drawing.

The procedure for attaching shadows using shadow polygons is as follows:

1. Set rendering order

Because both mask- and draw-shadow polygons must be translucent and must exist in the drawing order (see the "Cautions" on page 265), set the translucent polygons to manual sort with SwapBuffers and set them to be rendered in the order they are transmitted to the Geometry Engine.

↓

2. Draw the shadow polygon for the mask

Set the polygon attributes to [Draw only the back surface], [ID = 0], [α = 1 - 30] and [Shadow polygon] and draw the mask-shadow polygon. The rendering engine does not update the color buffer and creates only the mask images with 1 set in the stencil buffer.

↓

3. Draw the shadow polygon for rendering

Next, set the polygon attributes to [Draw both surfaces], [ID = 1–63], [α = 1–30] and [Shadow polygon] and draw the shadow polygons for rendering. The rendering engine first reads the stencil buffer, and if the value is 1, resets it to 0. If the value is 0, the engine attempts to draw to the color buffer.

- Polygon ID

  The draw-shadow polygon is drawn at this time only if its ID differs from both the ID of the opaque polygon and the ID of the translucent polygon in the Attribute buffer. This specification prevents an object from casting a shadow on itself by setting the same polygon ID for both the object and the draw-shadow polygon.

  When multiple shadows overlap, you can control whether to overlay them by setting them to the same polygon ID.

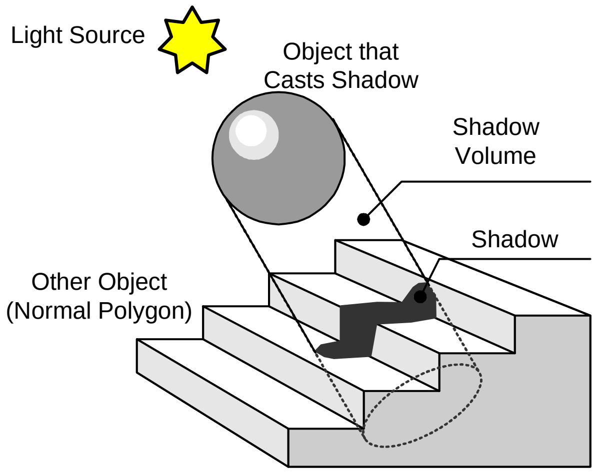
- Fog Enable Flag

  When a shadow polygon is rendered, the results of a logical AND operation applied to the fragment's fog enable flag and the Attribute buffer's fog enable flag are written to the Attribute buffer. With this feature, it is possible to exclude shaded areas from fog.

  (Example: A spotlight can be expressed by excluding an area from black fog.)

- Shadow Volume

Figure 7-33: Shadow Volume



- **Shadow-volume shape**

  In principle, the shadow volume takes the shape of a closed 3D shape. Note the "Cautions" on page 265 if you plan to use an open 3D shape in order to reduce the number of polygons.

- **Shadow-volume direction**

  To create the shadow of a spherical object, a cylinder-shaped shadow volume is created on the straight line defined by the light source and the spherical object.

- **Shadow-volume position**

  The cylinder-shaped shadow volume is located where it cannot be seen from the light source (a place not illuminated by the light of the sphere).

- **Shadow-volume length**

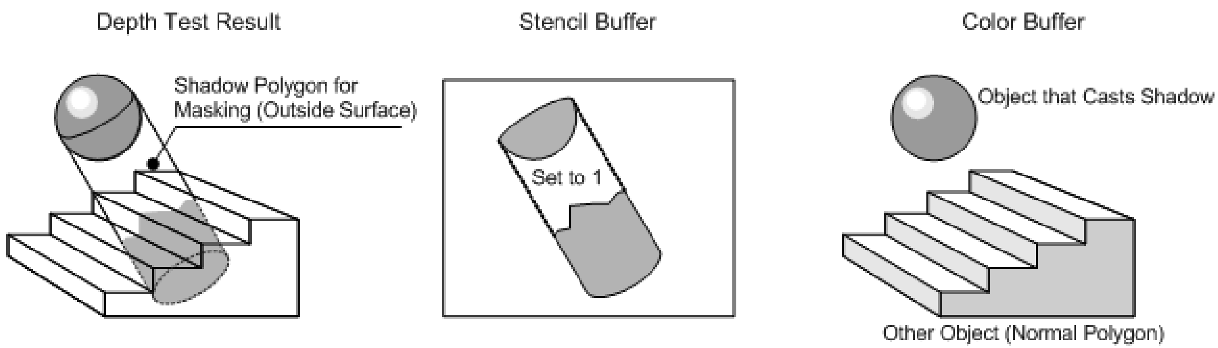
  The shadow is drawn on the surface of the object that is located inside the shadow volume. Thus, the length of the shadow volume should be long enough to pass through the surface of the object on which you want the shadow to be drawn.

Stencil buffer calculations

1. When drawing a shadow polygon for a mask

   The Stencil buffer is set to 1 when the depth test fails without affecting the Color buffer.

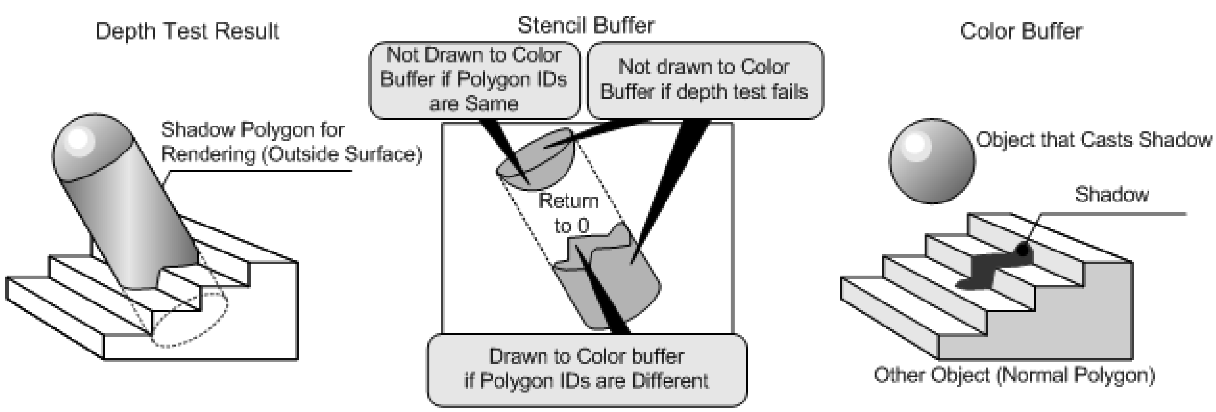
Figure 7-34: When Drawing a Shadow Polygon for a Mask



2. When drawing a shadow polygon for rendering

   If the Stencil buffer is already set to 1, it is reset to 0. When the depth test succeeds, if the polygon ID in the attribute does not match the polygon ID of the shadow polygon for rendering, the polygon is drawn to the color buffer.

Figure 7-35: When Drawing the Shadow Polygon for Rendering



As the diagrams illustrate, the shadow is drawn to a portion of one side of the shadow polygon for rendering. That is why you cannot achieve the effect of directly pasting textures to shadows when texture mapping.

Cautions

1. Order for rendering shadow polygons

To create a single shadow, draw the necessary shadow polygon for the mask and then the shadow polygon for rendering. If you draw a collection of shadow polygons for the mask and then draw the corresponding collection of shadow polygons for rendering, shadows may not be created in the intended regions.

Correct rendering string:

Shadow volume for mask 1 → Shadow volume for rendering 1 →
Shadow volume for mask 2 → Shadow volume for rendering 2

2. Shadow volumes with open shapes

When the shadow volume takes the shape of an open 3D shape, a region that has a shadow polygon for rendering but not a shadow polygon for the mask may be generated. This results in the creation of an incorrect shadow.

Note that when a shadow volume is clipped, it takes on an open shape.

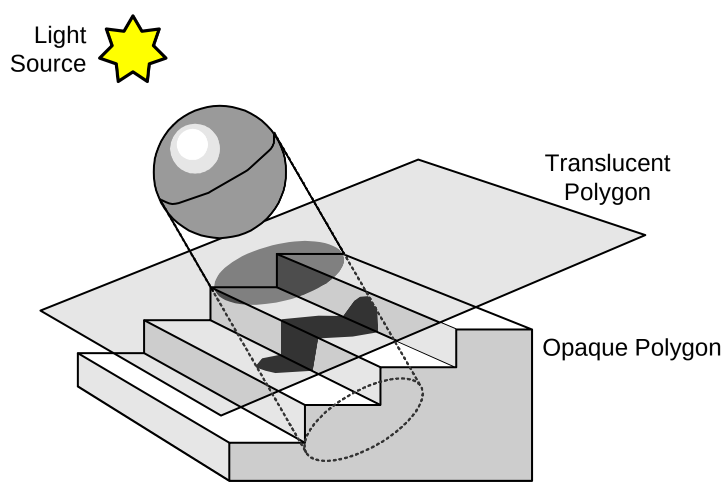
Technique

By following the procedure below, you can create scenes in which shadows are cast on translucent polygons. (However, this technique requires more shadow polygons.)

1. Opaque polygon →
2. Shadow polygon (shadow on opaque polygon) →
3. Translucent polygon on which a shadow is cast (update depth buffer) →
4. Shadow polygon (the shadow on the translucent polygon / polygon ID is the same as in step 2) →
5. Translucent polygon on which the shadow is not cast.

Figure 7-36 shows a schematic representation.

Figure 7-36: Technique for Rendering a Shadow on a Translucent Polygon



7.3.4.5 Toon Shading/Highlight Shading

ToonTable: Toon Table Register

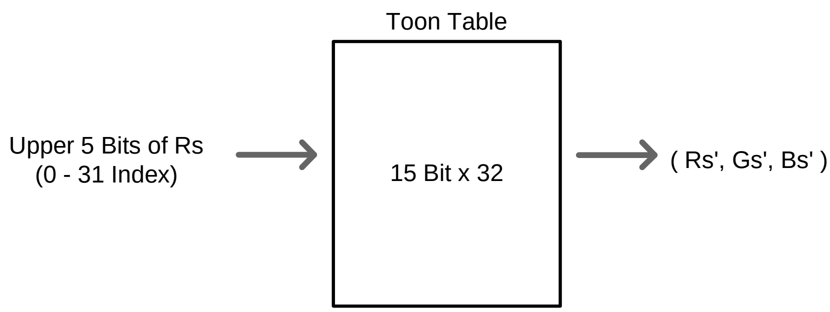
Name	Address	Attribute	Initial Value
TOON_TABLE_x (x=0–15)	0x04000380, 0x04000384, 0x04000388, 0x0400038C, 0x04000390, 0x04000394, 0x04000398, 0x0400039C, 0x040003A0, 0x040003A4, 0x040003A8, 0x040003AC, 0x040003B0, 0x040003B4, 0x040003B8, 0x040003BC	W	0x00000000

31	30–26	25–21	20–16	15	14–10	9–5	4–0
	BLUE n1	GREEN n1	RED n1		BLUE n0	GREEN n0	RED n0
	RGB transformation values when brightness n1=2x+1				RGB transformation values when brightness n0=2x		

This register is used for toon shading and highlight shading.

In toon shading and highlight shading, the fragment color R value is treated as the brightness, and this R value (upper 5 bits) is used as the index to reference RGB values from the toon table.

Figure 7-37: Transformations Using a Toon Table



7.3.4.5.1 Toon Shading

The fragment color R value is treated as the brightness, and this R value (the upper 5 bits) is used as the index to reference RGB values from the toon table to set the new fragment colors.

The texture color and the post-table-reference fragment color both have (R_5:G_5:B_5 = 5:5:5) number of bits. Therefore, before the texture-blending computation is conducted, the following formulas are used to expand the number of bits to (R_6:G_6:B_6 = 6:6:6).

$R_6 = R_5 << 1$ (When R_5 is 0)

$R_6 = (R_5 << 1) + 1$ (When R_5 is nonzero)

Texture mapping involves the same equations used in Modulation mode, except that the toon table–transformed values are used for the fragment colors (see Table 7-12).

Table 7-12 : Texture Blending Equations (toon table)

Type of Texture	Translucent Texture	Non-translucent Texture
Texture Blending Equations	R = { (Rt+1) x (Rs'+1) -1} / 64 G = { (Gt+1) x (Gs'+1) -1} / 64 B = { (Bt+1) x (Bs'+1) -1} / 64 A = { (At+1) x (As+1) -1} / 64	R = { (Rt+1) x (Rs'+1) -1} / 64 G = { (Gt+1) x (Gs'+1) -1} / 64 B = { (Bt+1) x (Bs'+1) -1} / 64 A = At x As

(R, G, B, A) : Newly written fragment color (fractional parts resulting from calculations are truncated)

(Rt, Gt, Bt, At) : Texture color expanded to (R:G:B = 6:6:6) bits

(Rs, Gs, Bs, As) : Fragment color expanded to (R:G:B = 6:6:6) bits

(Rs', Gs', Bs') : Fragment color expanded to (R:G:B = 6:6:6) bits after table conversion

Note: The toon table is referenced for fragment colors shared by all toon-shaded polygons. If you want every toon shading polygon to have different colors, use textures to color even monochrome polygons.

Using ambient reflection color and emission color to raise the minimum value of the fragment color R value also narrows the effective range of R, and this coarsens the gradation of toon shading. Because material color loses its meaning when toon shading, we recommend that you use only diffusion reflection color so as to retain gradation.

7.3.4.5.2 Highlight Shading

The fragment color R value is treated as the brightness, and this R value (the upper 5 bits) is used as the index to reference RGB values (color offset values) from the toon table to add to the fragment color.

The texture color and the post-table-reference fragment color both have (R_5:G_5:B_5 = 5:5:5) number of bits. Therefore, before the texture-blending computation is conducted, the following formula is used to extend the number of bits to (R_6:G_6:B_6 = 6:6:6).

$R_6 = R_5 << 1$ (If R_5 is 0)

$R_6 = (R_5 << 1) + 1$ (If R_5 is not 0)

When texture mapping, a color offset value is added to texture colors whose RGB values have each been modulated by the fragment color’s R value (see Table 7-13). This can produce an effect as if the texture is highlighted (emitting a color brighter than the texture's own color).

Table 7-13 : Texture Blending Equation (Highlight Shading)

Texture Type	Translucent Texture	Non-translucent Texture
Texture Blending Equations	R = min[63, { (Rt+1) x (Rs+1) -1} / 64 + Rs’] G = min[63, { (Gt+1) x (Rs+1) -1} / 64 + Gs’] B = min[63, { (Bt+1) x (Rs+1) -1} / 64 + Bs’] A = { (At+1) x (As+1) -1} / 32	R = min[31, { (Rt+1) x (Rs+1) -1} / 64 + Rs’] G = min[31, { (Gt+1) x (Rs+1) -1} / 64 + Gs’] B = min[31, { (Bt+1) x (Rs+1) -1} / 64 + Bs’] A = At x As

(R, G, B, A) : Newly written fragment color (fractional parts resulting from calculations are truncated)

(Rt, Gt, Bt, At) : Texture color extended to (R:G:B = 6:6:6) bits

(Rs, Gs, Bs, As) : Fragment color

(Rs’, Gs’, Bs’) : Fragment color extended to (R:G:B = 6:6:6) bits after table conversion

Note: Because a color offset is added, sometimes the hue of the fragment color changes.

7.3.5 Textures

7.3.5.1 Texture Blending

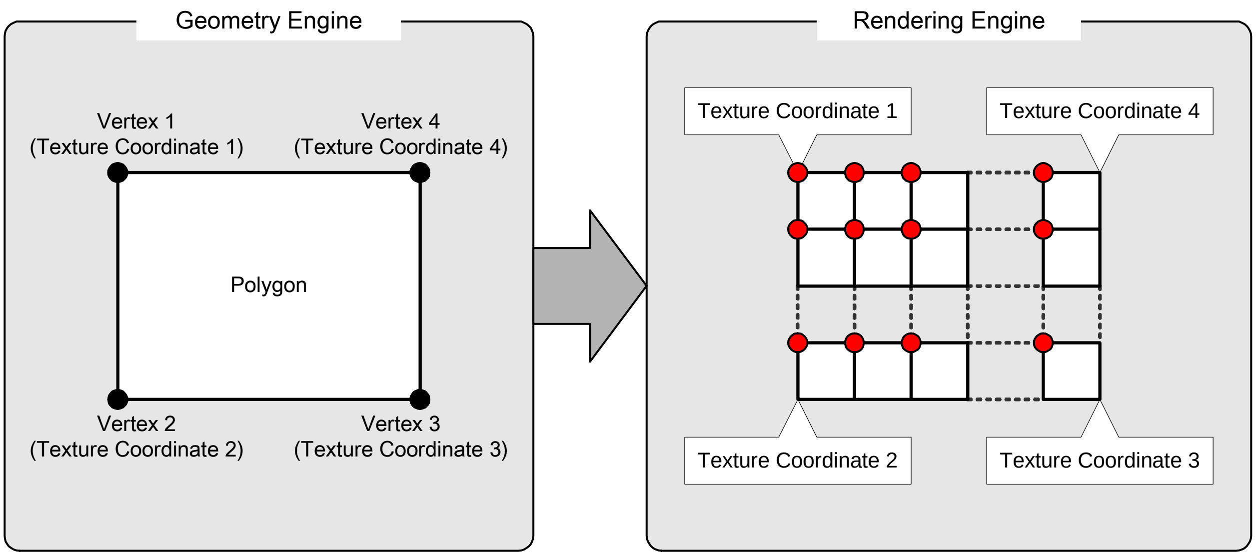
The Rendering Engine interpolates the texture coordinates corresponding to each pixel inside the polygon using the texture settings (flip/repeat) passed by the Geometry Engine, the vertex texture coordinates, and the *w* value.

The process of blending texel color with the color of each pixel of the polygon is called texture blending. The mode for this texture blending can be set to either Decal or Modulation, using the PolygonAttr register.

7.3.5.1.1 Texture Image Sampling

In TWL, the texture image is sampled by interpolating, from each vertex's texture coordinates, texture coordinates that correspond to the top-left of each pixel in a polygon (see Figure 7-38).

Figure 7-38: Texture Image Sampling

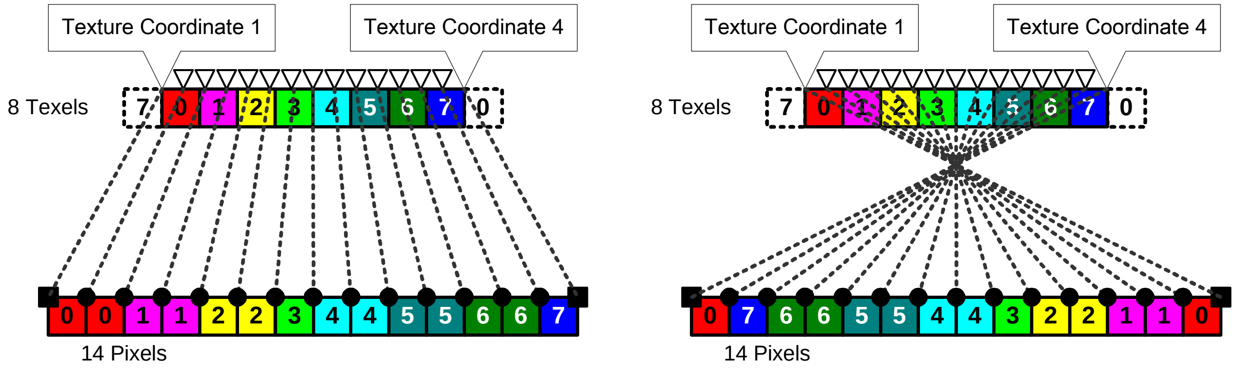


For example, if a polygon has 8x8 textures applied to it and its display width is 14 dots, pixel and texel correspondence are as shown on the left in Figure 7-39.

If a polygon is turned front to back like this, the texel/pixel correspondence slips.

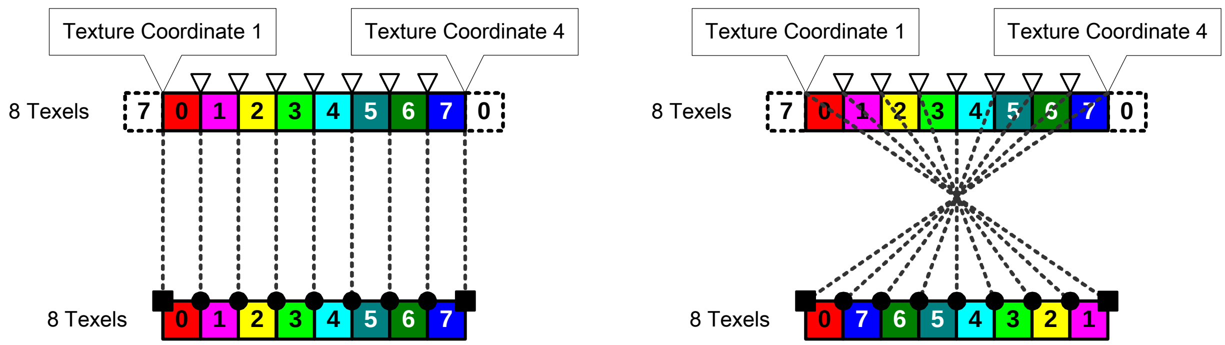
In the diagram on the right of Figure 7-39, texel 0 appears at the left edge (if the texture is not H-flipped, it is omitted). On the right edge, texel 0 is only sampled once.

Figure 7-39: Applying an 8x8 texel Texture to an 14-dot Wide Polygon



If a polygon has 8x8 textures applied to it and its display width is 8 dots, pixel and texel correspondence are as shown in Figure 7-40.

Figure 7-40: Applying an 8x8 texel Texture to an 8-dot Wide Polygon



When using polygons for 2D displays such as OBJ or BG, texels and pixels have a one-to-one correspondence. However, in a case such as this, when rendering a polygon that is mapped with an 8x8 texel texture on 8x8 pixels of an LCD, the front and back surfaces are displayed as shown in Figure 7-41.

Figure 7-41: Displaying Front and Back Surfaces of an LCD

Front Surface

00	01	02	03	04	05	06	07
10	11	12	13	14	15	16	17
20	21	22	23	24	25	26	27
30	31	32	33	34	35	36	37
40	41	42	43	44	45	46	47
50	51	52	53	54	55	56	57
60	61	62	63	64	65	66	67
70	71	72	73	74	75	76	77

Horizontal Reflection of Polygons

00	07	06	05	04	03	02	01
10	17	16	15	14	13	12	11
20	27	26	25	24	23	22	21
30	37	36	35	34	33	32	31
40	47	46	45	44	43	42	41
50	57	56	55	54	53	52	51
60	67	66	65	64	63	62	61
70	77	76	75	74	73	72	71

Vertical Reflection of Polygons

00	01	02	03	04	05	06	07
70	71	72	73	74	75	76	77
60	61	62	63	64	65	66	67
50	51	52	53	54	55	56	57
40	41	42	43	44	45	46	47
30	31	32	33	34	35	36	37
20	21	22	23	24	25	26	27
10	11	12	13	14	15	16	17

90 Degree Clockwise Rotation
(270 Degrees Counterclockwise)

00	70	60	50	40	30	20	10
01	71	61	51	41	31	21	11
02	72	62	52	42	32	22	12
03	73	63	53	43	33	23	13
04	74	64	54	44	34	24	14
05	75	65	55	45	35	25	15
06	76	66	56	46	36	26	16
07	77	67	57	47	37	27	17

180 Degree Rotation

00	07	06	05	04	03	02	01
70	77	76	75	74	73	72	71
60	67	66	65	64	63	62	61
50	57	56	55	54	53	52	51
40	47	46	45	44	43	42	41
30	37	36	35	34	33	32	31
20	27	26	25	24	23	22	21
10	17	16	15	14	13	12	11

270 Degree Clockwise Rotation
(90 Degrees Counterclockwise)

00	10	20	30	40	50	60	70
07	17	27	37	47	57	67	77
06	16	26	36	46	56	66	76
05	15	25	35	45	55	65	75
04	14	24	34	44	54	64	74
03	13	23	33	43	53	63	73
02	12	22	32	42	52	62	72
01	11	21	31	41	51	61	71

7.3.5.1.2 Decal Mode

Depending on the texture's α value, either the result of Gouraud shading of the vertex color created by the lighting process (fragment color) or the texture's color value is displayed.

For translucent textures, blending is done with the texture's α value.

Table 7-14 shows the texture-blending expression used in decal mode.

The texture color has (R_5:G_5:B_5 = 5:5:5) number of bits. Therefore, before the texture-blending computation is conducted, the following formulas are used to expand the number of bits to (R_6:G_6:B_6 = 6:6:6).

$R_6 = R_5 << 1$ (When R_5 is 0)

$R_6 = (R_5 << 1) + 1$ (When R_5 is non-zero)

Table 7-14 : Texture Blending Equations (Decal Mode)

Texture Type	Translucent Texture	Non-translucent Texture
Texture Blending Equations	R = {At x Rt + (31 - At) Rs} / 32 G = {At x Gt + (31 - At) Gs} / 32 B = {At x Bt + (31 - At) Bs} / 32 A = As Handling exceptions: When At = 0, (R, G, B, A) = (Rs, Gs, Bs, As) is used. When At =31, (R, G, B, A) = (Rt, Gt, Bt, As) is used.	R = At x Rt + (1 - At) Rs G = At x Gt + (1 - At) Gs B = At x Bt + (1 - At) Bs A = As

(R, G, B, A) : Newly written fragment color (fractional parts resulting from calculations are truncated)

(Rt, Gt, Bt, At) : Texture color expanded to (R:G:B = 6:6:6) bits

(Rs, Gs, Bs, As) : Fragment color

7.3.5.1.3 Modulation Mode

The result of the Gouraud shading of the vertex color (fragment color) created by the lighting process is modulated by the texture's color value and displayed.

Table 7-15 shows the texture-blending expressions used in modulation mode.

The fragment colors of texture colors have (R_5:G_5:B_5 = 5:5:5) number of bits. Therefore, before the texture-blending computation is conducted, the following formulas are used to expand the number of bits to (R_6:G_6:B_6 = 6:6:6).

$R_6 = R_5 << 1$ (When R_5 is 0)

$R_6 = (R_5 << 1) + 1$ (When R_5 is nonzero)

Table 7-15 : Texture Blending Expressions (Modulation Mode)

Type of Texture	Translucent Texture	Non-Translucent Texture
Texture Blending Equations	R = { (Rt+1) x (Rs+1) -1} / 32 G = { (Gt+1) x (Gs+1) -1} / 32 B = { (Bt+1) x (Bs+1) -1} / 32 A = { (At+1) x (As+1) -1} / 32	R = { (Rt+1) x (Rs+1) -1} / 32 G = { (Gt+1) x (Gs+1) -1} / 32 B = { (Bt+1) x (Bs+1) -1} / 32 A = At x As

(R, G, B, A) : Newly written fragment color (fractional parts resulting from calculations are truncated)

(Rt, Gt, Bt, At) : Texture color expanded to (R:G:B = 6:6:6) bits

(Rs, Gs, Bs, As) : Fragment color

7.3.5.2 Texture Formats

TWL can handle seven different texture formats. Table 7-16 lists the texture formats.

Table 7-16 : List of Texture Formats

Format	Number of Selectable Colors for 1 Texel	Palette Base Boundary (see note)	α Value Bits	Number of Bits per Texel
4-Color Palette Texture	4	0x08	0	2
16-Color Palette Texture	16	0x10	0	4
256-Color Palette Texture	256	0x10	0	8
4x4 Texel Compressed Texture	4 (every 4x4 texels)	0x10	0	3 (Includes palette index data)
A3I5 Translucent Texture	32	0x10	3	8
A5I3 Translucent Texture	8	0x10	5	8
Direct Color Texture	32,768	Palette not used	1	16

Note: *Palette Base Boundary* is the amount by which the address is increased when the palette base is increased by 1 by the `TexPlttBase` command

7.3.5.2.1 Texture Images

The Rendering Engine references the texture image slot's texture image in the format specified by the `TexImageParam` command. The texture image is composed of texel data.

7.3.5.2.1.1 4-Color Palette Textures

Texel Data Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T7		T6		T5		T4		T3		T2		T1		T0	
8 Texels of Data (2 bits / texel)															

- T7–T0 : Texel Data

  Specifies the texture color palette color number (0–3)

Display Texture

T0	T1	T2	T3	T4	T5	T6	T7

7.3.5.2.1.2 16 Color Palette Textures

Texel Data Format

15 12	11 8	7 4	3 0
T3	T2	T1	T0
4 Texels of Data (4 bits / texel)			

- T3–T0 : Texel Data

  Specifies the texture color palette color number (0–15).

Display Texture

T0	T1	T2	T3				

7.3.5.2.1.3 256-Color Palette Textures

Texel Data Format

15 8	7 0
T1	T0
2 Texels of Data (8 bits / texel)	

- T1–T0 : Texel Data

  Specifies the texture color palette color number (0–255).

Display Texture

T0	T1						

7.3.5.2.1.4 4x4 Texel Compression Textures

This format can obtain the compression effect by dividing the image into 4x4 pixel blocks and then converting them to images with palettes with 2-bit indexes.

Texel Data Format

31			24	23			16	15			8	7			0
T33	T32	T31	T30	T23	T22	T21	T20	T13	T12	T11	T10	T03	T02	T01	T00
4 x 4 Texels of Data (2 bits / texel)															

- T33–T00 : Texel Data

  Specifies the color number (0–3).

Display Texture

T00	**T01**	**T02**	**T03**				
T10	**T11**	**T12**	**T13**				
T20	**T21**	**T22**	**T23**				
T30	**T31**	**T32**	**T33**				

Palette Index Data Format

15	14	13					8	7							0
A	PTY														
Palette Settings		Palette Address													

- Palette settings
  - A : 3 colors / 4 colors setting flag

0	3 colors + Transparent mode (Color 3 is transparent color)
1	4-color mode

  - PTY : Palette type selection flag

0	4-color palettes (Four palettes for each 4x4 texels)
1	Linear interpolation 4-color palettes (Two palettes for each 4x4 texels)

- Palette address

The address of the texture palette slot in which color data is stored is specified in units of 2 colors (units of 4 bytes).

Color 0 is the color located at the texture palette slot address, calculated as follows:

(The value set by the TexPlttBase command x 0x10) + (the palette address setting value x 4)

The texel color value RGB components are calculated as shown in Table 7-17.

Table 7-17 : Texel Color Values

		PTY=0
PTY=0	**A=0**	Color 0[5:0]:(Palette 0[4:0]==0) ? (Palette 0[4:0] x 2) : (Palette 0[4:0] x 2 + 1) Color 1[5:0]:(Palette 1[4:0]==0) ? (Palette 1[4:0] x 2) : (Palette 1[4:0] x 2 + 1) Color 2[5:0]:(Palette 2[4:0]==0) ? (Palette 2[4:0] x 2) : (Palette 2[4:0] x 2 + 1) Color 3[5:0]:Transparent Color
	A=1	Color 0[5:0]:(Palette 0[4:0]==0) ? (Palette 0[4:0] x 2) : (Palette 0[4:0] x 2 + 1) Color 1[5:0]:(Palette 1[4:0]==0) ? (Palette 1[4:0] x 2) : (Palette 1[4:0] x 2 + 1) Color 2[5:0]:(Palette 2[4:0]==0) ? (Palette 2[4:0] x 2) : (Palette 2[4:0] x 2 + 1) Color 3[5:0]:(Palette 3[4:0]==0) ? (Palette 3[4:0] x 2) : (Palette 3[4:0] x 2 + 1)
PTY=1	**A=0**	Color 0[5:0]:Palette 0[4:0] x 2 Color 1[5:0]:Palette 1[4:0] x 2 Color 2[5:0]:Palette 0[4:0] + Palette 1[4:0] Color 3[5:0]:Transparent Color
	A=1	Color 0[5:0]:Palette 0[4:0] x 2 Color 1[5:0]:Palette 1[4:0] x 2 Color 2[5:0]:(Palette 0[4:0] x 5 + Palette 1[4:0] x 3) / 4 Color 3[5:0]:(Palette 0[4:0] x 3 + Palette 1[4:0] x 5) / 4

One set of palette index data is associated with each 4x4 texel (see "Figure 7-42 : Texture Image Slots" on page 277).

Texture Image Slots

In 4x4 texel compression mode, the texture image data and the texture palette index data should be mapped as follows:

- Texel data

  Map to texture image slots 0 and 2.

- Texture palette index data

  Map to texture image slot 1.

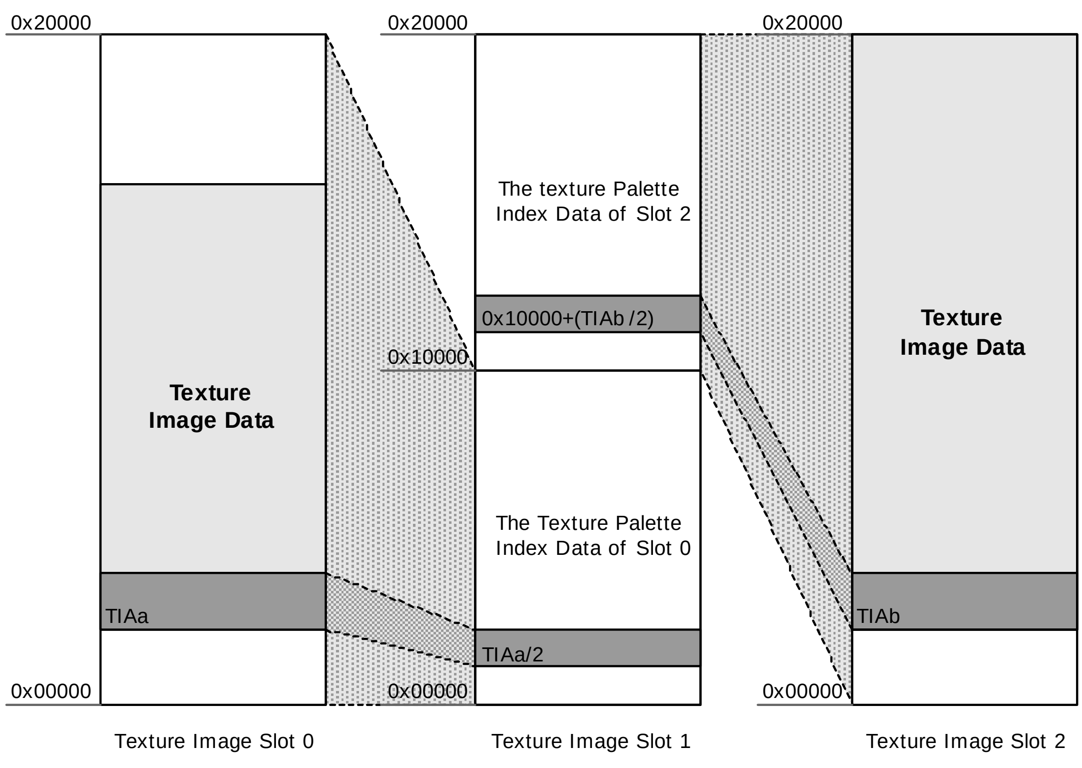
- Correspondence between texel data and texture palette index data

  The texture palette index data that corresponds to the 4x4 texel data in the TIAa address of texture image slot 0 should be placed in the (TIAa/2) address of texture image slot 1.

  The texture palette index data that corresponds to the 4x4 texel data in the TIAb address of texture image slot 2 should be placed in the (0x10000 + (TIAb/2)) address of texture image slot 1. (See Figure 7-42.)

  Each 4x4 texel is associated with one set of texture palette index data. Individual texel colors take the color of the address (the texture color palette slot) specified by the texture palette index data as 0 and use the color number's color designated by the texel data.

Figure 7-42 : Texture Image Slots



Note: Texture data is placed in slot 0 and slot 2. Because the address jumps, it is not possible to reference 1024x1024. In 4x4-texel compressed-texture mode, use a texture size that is no larger than 1024x512.

7.3.5.2.1.5 A3I5 Translucent Textures

Texel Data Format

15 13	12 8	7 5	4 0
ALPHA	INDEX	ALPHA	INDEX
T1		T0	

- T1–T0 : Texel data
  - ALPHA : α-value

    Specifies that the degree of transparency for the texel. 0 is Transparent.

    The α–value is used in the Rendering Engine, extended in 5 bits as shown in the table below.

    (5-bit α = {(3 bit α << 2) + (3 bit α >> 1)})

3 bit α	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
5 bit α	0	4	9	13	18	22	27	31

  - INDEX : Color number

    Specifies the color number (0–31) of the texture color palette.

Display Texture

T0	T1						

7.3.5.2.1.6 A5I3 Translucent Textures

Texel Data Format

15 11	10 8	7 3	2 0
ALPHA	INDEX	ALPHA	INDEX
T1		T0	

- T1–T0 : Texel data
  - ALPHA : α-value

    Specifies that the degree of transparency for the texel. 0 is Transparent.
  - INDEX : Color number

    Specifies the color number (0–7) of the texture color palette.

Display Texture

T0	**T1**						

7.3.5.2.1.7 Direct Color Textures

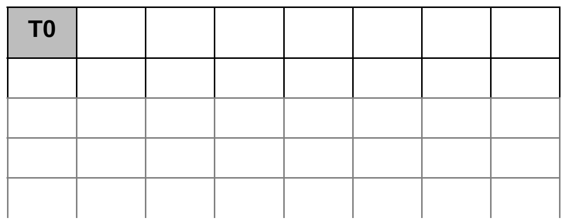
Texel Data Format

15	14–10	9–5	4–0
ALPHA	BLUE	GREEN	RED
T0			

- T0 : Texel data

  Directly configures texel color. The texture color palette is not used.

Display Texture



7.3.5.2.2 Texture Palette

The texture palette slot stores the texture color data.

Texture Color Data Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	TEX_COLOR_BLUE					TEX_COLOR_GREEN					TEX_COLOR_RED				
	Color														

The texture format determines the way in which texture color palettes are referenced. (The texture formats are: 4-color, 16-color, 256-color, 4x4 compressed-texel, A3I5, and A5I3.)

Generally, the color number is referenced for the color determined by the texel data from the palette specified by the texture palette base. However, when the texture is a 4x4 compressed-texel-format texture, the palette is specified by setting the palette address, in addition to the texture palette base. See Figure 7-43 to Figure 7-46 for details on the palette base for each texture color palette and palette address mapping.

Figure 7-43: Palette Base and Palette Address (4-color palette)

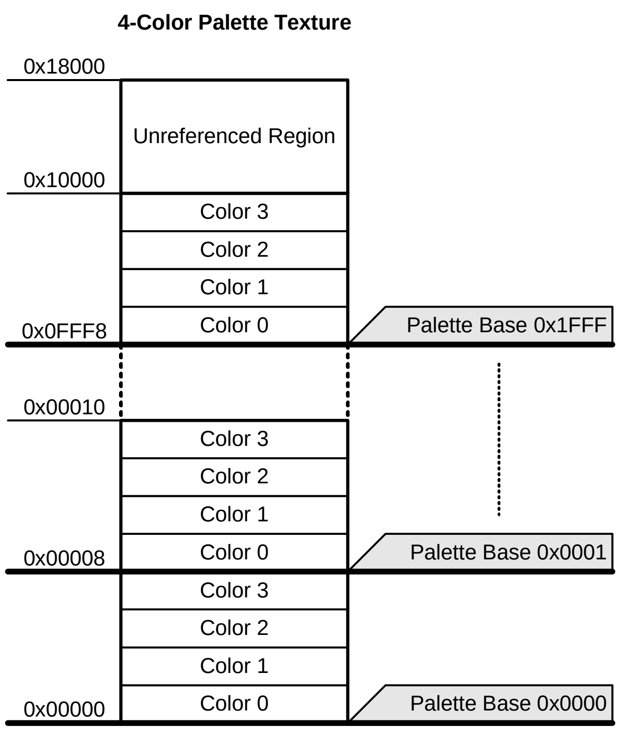


Figure 7-44: Palette Base and Palette Address (16-Color Palette and 256-Color Palette)

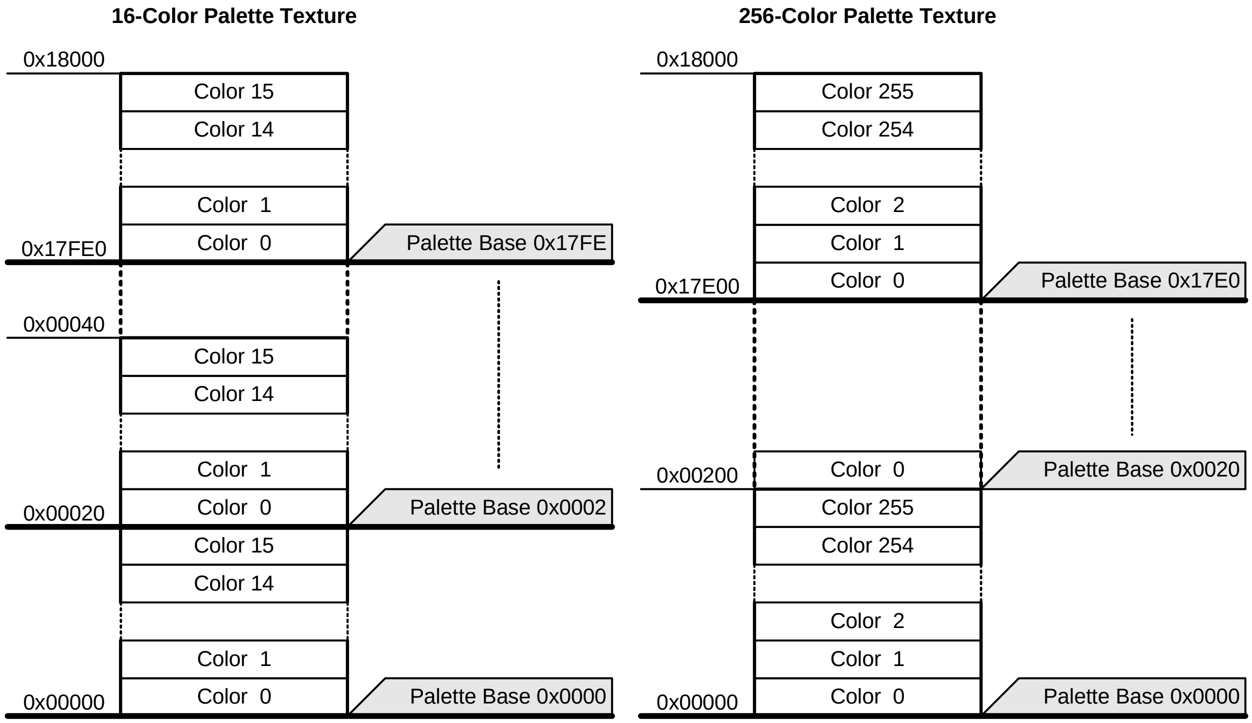
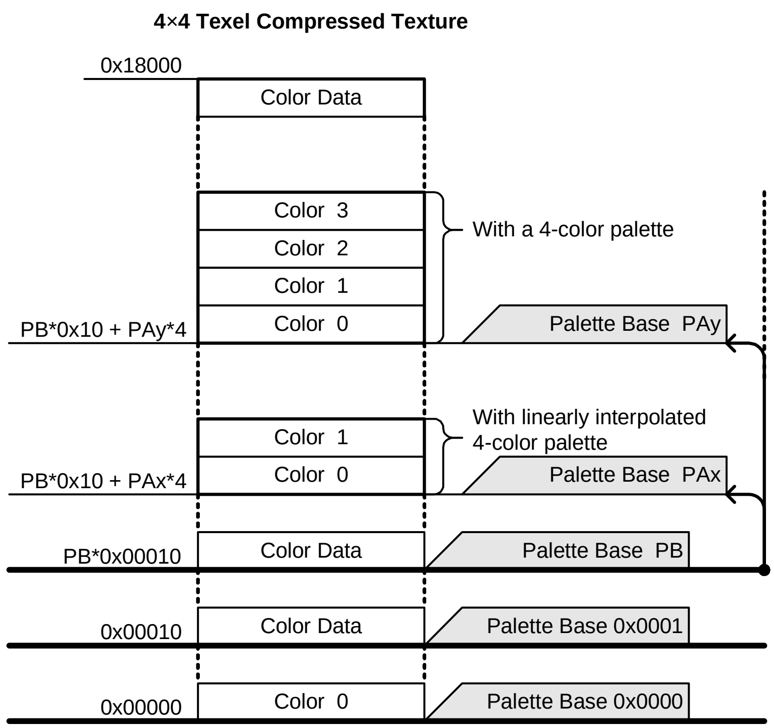


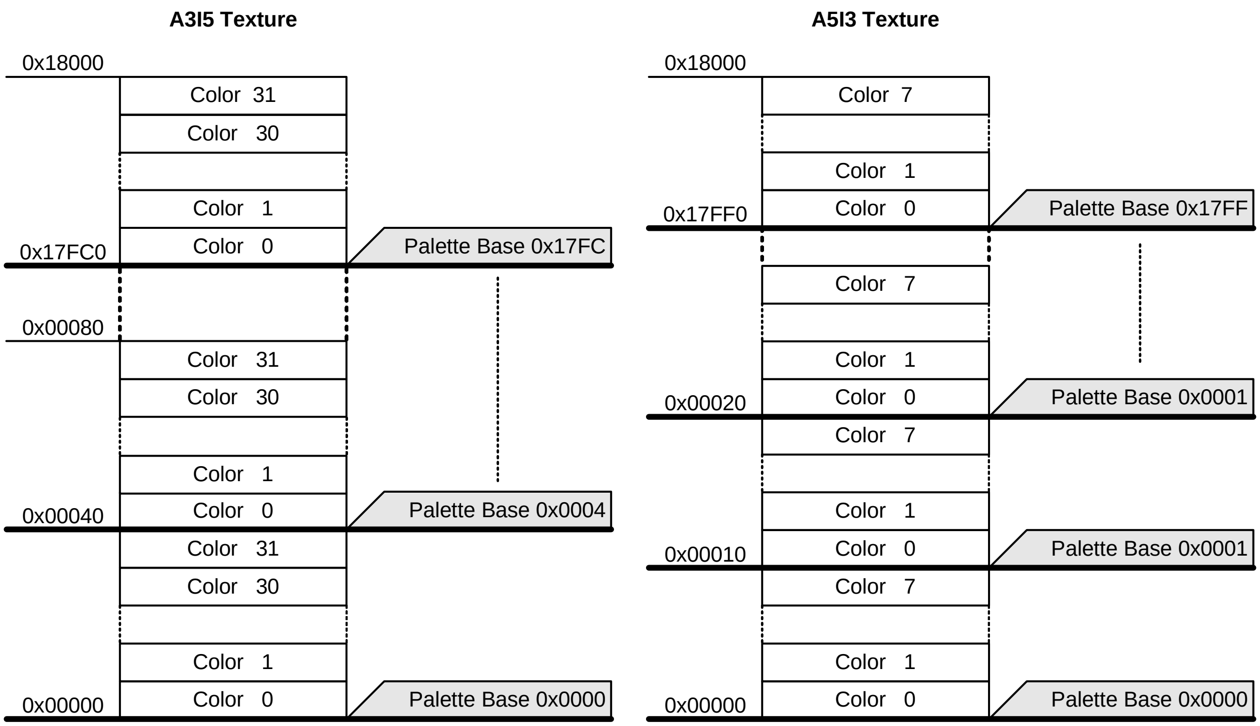
Figure 7-45: Palette Base and Palette Address (4x4 Texel Compression)



For a 4x4 compressed-texel-format texture, the palette address is set by the palette base and the palette address. Color numbers 0 to 3 are referenced for a four-color palette, but only color numbers 0 and 1 are referenced for a linear interpolated four-color palette. The remaining two colors applied to the texels are calculated from Color 0 and Color 1.

Even for a four-color palette, Transparent is used without referencing Color 3 when the mode is set to Three Colors + Transparent.

Figure 7-46: Palette Base and Palette Address (A3I5, A5I3)



7.3.6 Alpha-Test

After texture blending, the fragment's alpha value is compared with the value set in the AlphaReference register. Rendering does not occur when the α value is less than this reference value.

AlphaReference: Alpha-Test Comparison Value Register

Name: ALPHA_TEST_REF Address: 0x04000340 Attribute: W Initial value: 0x0000

15							8	7			4 0
											ALPHA_REFERENCE
											α-Test Comparison Value

- ALPHA_REFERENCE[d04–d00] : α-Test comparison value

  When α-Test is set to ON, pixels that have an α value below this specified value are not drawn.

- Wireframe display polygons

  Setting α = 0 in the polygon attributes displays polygons as wireframes—α no longer retains its original meaning.

  If translucent textures are unmapped, the wireframe section actually has an α value of 31.

  Therefore, if the α test reference level is set when the α test is ON when displaying a wireframe, it is displayed for any reference value other than 31.

7.3.7 Alpha-Blending

The specifications call for the Rendering Engine to perform alpha-blending by first creating the 3D screen before alpha-blending with the 2D screen. Alpha-blending with the 2D screen is performed using the 2D graphics color special effect functions.

You can control the Rendering Engine's alpha-blending process by setting the DISP3DCNT register's α-blending enable flag on/off.

Table 7-18 shows the equation used when alpha-blending.

Table 7-18: Equation when α-Blending

α-Blending Enable Flag	Calculations for Newly Stored Data in Color Buffer
When ON	R = {(As + 1) x Rs + (31 - As) x Rb} / 32 G = {(As + 1) x Gs + (31 - As) x Gb} / 32 B = {(As + 1) x Bs + (31 - As) x Bb} / 32 A = max[As, Ab] Handling exceptions: When As = 0, (R, G, B, A) = (Rb, Gb, Bb, max[As, Ab]) is used. When As = 31 or Ab = 0, (R, G, B, A) = (Rs, Gs, Bs, max[As, Ab]) is used.
When OFF	When As = 0 (R, G, B, A) = (Rb, Gb, Bb, Ab) When As is non-zero (R, G, B, A) = (Rs, Gs, Bs, As)

(R, G, B, A) : Newly written fragment color (fractional parts resulting from calculations are truncated)

(Rb, Gb, Bb, Ab) : Color buffer's color

(Rs, Gs, Bs, As) : Fragment color

7.3.7.1 3D Alpha-Blending

The color buffer's color value and the fragment's color value are blended based on the fragment's α value, and the result is then written back to the color buffer.

7.3.7.2 2D and 3D Alpha-Blending Preprocess

In the 3D alpha-blending process described above, the color buffer's α value is updated only when it is smaller than the fragment's α value. This specification is set in order to approximate the α value in regions where translucent polygons overlap when alpha-blending 2D and 3D.

When the color buffer's alpha value is 0 and translucent polygons are drawn, the fragment's color is written to the color buffer without any alpha-blending.

You can thus achieve a more natural composite with the 2D screen by using ClearColor to zero-clear the color buffer's alpha value, since the ClearColor's RGB values are not blended.

Refer to "7.4.4.1 Alpha-Blending with the 2D Screen" on page 295 for details on this subject.

7.3.8 Edge Marking

Edge marking is a feature for outlining the edges of opaque polygons with different polygon IDs in the Attribute buffer. You can control this feature by setting the DISP3DCNT register's edge-marking enable flag on or off.

The colors that are used in edge marking are the eight colors that are selected using the upper 3 bits of the polygon ID as an index.

EdgeColor: Edge Color Register

Name	Address	Attribute	Initial Value
EDGE_COLOR_x (x =0-3)	0x04000330, 0x04000334, 0x04000338, 0x0400033C	W	0x00000000

<table>
<tr><td>31</td><td>30 … 26</td><td>25 24 | 23 … 21</td><td>20 … 16</td><td>15</td><td>14 … 10</td><td>9 8 | 7 … 5</td><td>4 … 0</td></tr>
<tr><td></td><td>BLUE n1</td><td>GREEN n1</td><td>RED n1</td><td></td><td>BLUE n0</td><td>GREEN n0</td><td>RED n0</td></tr>
<tr><td></td><td colspan="3">Edge marking color when polygon ID (n1)=2x+1</td><td></td><td colspan="3">Edge marking color when polygon ID (n)=2x</td></tr>
</table>

- [d30–d16], [d14–d00] : Edge marking colors

  Specifies the eight colors to employ for edge marking.

  If a polygon is clipped, edge marking is also applied to the *clipping boundary* (a new edge created by clipping) A comparison to clear polygon ID is also made at the edge of the screen. Therefore, if the clear polygon ID and the polygon ID are the same, edge marking is not applied to the new edges created by clipping.

  Note: Edge marking of opaque polygons in the background behind translucent polygons sometimes does not work correctly when the PolygonAttr command has set the Translucent Polygon Depth Value Update–enable flag to 1 (because the Depth buffer's value is referenced when determining edge marking).

7.3.9 Fog Blending

Fog blending is the process of blending each pixel color with fog color in proportion to the depth value. The fog-blending process can create a fog effect on the screen, hiding from the player the far-plane clipping-and-rejection processes that do not arise in the natural world. You can control this feature by setting the DISP3DCNT register's fog-enable flag on or off.

7.3.9.1 3D Fog

In OpenGL, the fog process is performed on each polygon when it is drawn. But with TWL, the fog process is performed on the color buffer in which all polygons have been drawn after the edge marking process.

Because of this specification, you can produce effects in which fog is applied to everything except a specified region. However, applying fog to regions in which translucent polygons are rendered can produce unnatural effects. If this happens, use the `PolygonAttr` command to set the Translucent Polygon Depth Value Update–enable flag for visual approximation.

Note: If you update the depth value, you must also conduct a Z sort, or else some parts inside regions in which translucent polygons overlap may not be rendered.

FogColor: Fog Color Register

Name: FOG_COLOR Address: 0x04000358 Attribute: W Initial value: 0x00000000

31–24	23–21	20–16	15	14–10	9–5	4–0
		FOG_ALPHA		FOG_BLUE	FOG_GREEN	FOG_RED
		Fog α Value		Fog Color		

This specifies the fog color.

When compositing with the 2D screen's background, you can achieve the effect of the object's dissolving into the 2D screen if you set the fog α value to 0. This effect appears more natural if you set the Fog mode of the 3D Display Control register (DISP3DCNT) to Fog Blending Using only the Pixel's α Value.

FogOffset: Fog Offset Register

Name: FOG_OFFSET Address: 0x0400035C Attribute: W Initial value: 0x0000

15	14–0
	FOG_OFFSET
	Fog Offset

- FOG_OFFSET[d14–d00] : Fog offset

  Sets the depth value that is the basis for fog density calculations.

  The fog density for pixels that have an upper 15-bit depth value nearer than (fog offset + (0x400 >> fog shift)) is fixed to the fog density table's DENSITY0 value. The fog density for pixels that have a depth value farther than (fog offset + (0x400 >> fog shift) x 32) is fixed to DENSITY31.

  The fog shift is a value set by the DISP3DCNT register (the 3D Display Control register).

  If the depth value is within the two areas mentioned above, the fog density is the value that results from linear interpolation of two elements in the fog table.

  Because the depth value differs depending on the method of depth buffering (Z buffering or W buffering), the way in which fog is applied also differs. In short, the fog density depends on the depth-value curve to the z value of the View coordinates. To learn more about this relationship, see "7.2.5 Depth Buffering" on page 187. The method for depth buffering is selected using the Geometry Engine's `SwapBuffers` command.

FogTable: Fog Density Table Register

Name	Address	Attribute	Initial Value
FOG_TABLE_x (x=0-7)	0x04000360, 0x04000364, 0x04000368, 0x0400036C, 0x04000370, 0x04000374, 0x04000378, 0x0400037C	W	0x00000000

31	30–24	23	22–16	15	14–8	7	6–0
	DENSITY n3		DENSITY n2		DENSITY n1		DENSITY n0
	Fog Density n3=4x+3		Fog Density n2=4x+2		Fog Density n1=4x+1		Fog Density n0=4x

Specifies a 32-level fog density table.

The fog density for each pixel is the value that results from linear interpolation of the corresponding depth buffer value. You can thus approximate any fog density curve (see "Figure 7-47 : Depth Values and Fog Density" on page 287).

7.3.9.1.1 Fog Density Equations

Assume that F_IVL = (0x400 >> FOG_SHIFT) and that f(i) is the i^{th} parameter in the fog density table. Then the fog density and the upper 15-bit depth value (Zd) have the following relationship:

1. When 0x0000 ≤ Zd ≤ (FOG_OFFSET + F_IVL – 1):

   Fog density f = f (0)

2. When (FOG_OFFSET + F_IVL x i) ≤ Zd ≤ (FOG_OFFSET + F_IVL x (i + 1) - 1) (i = 1–31):

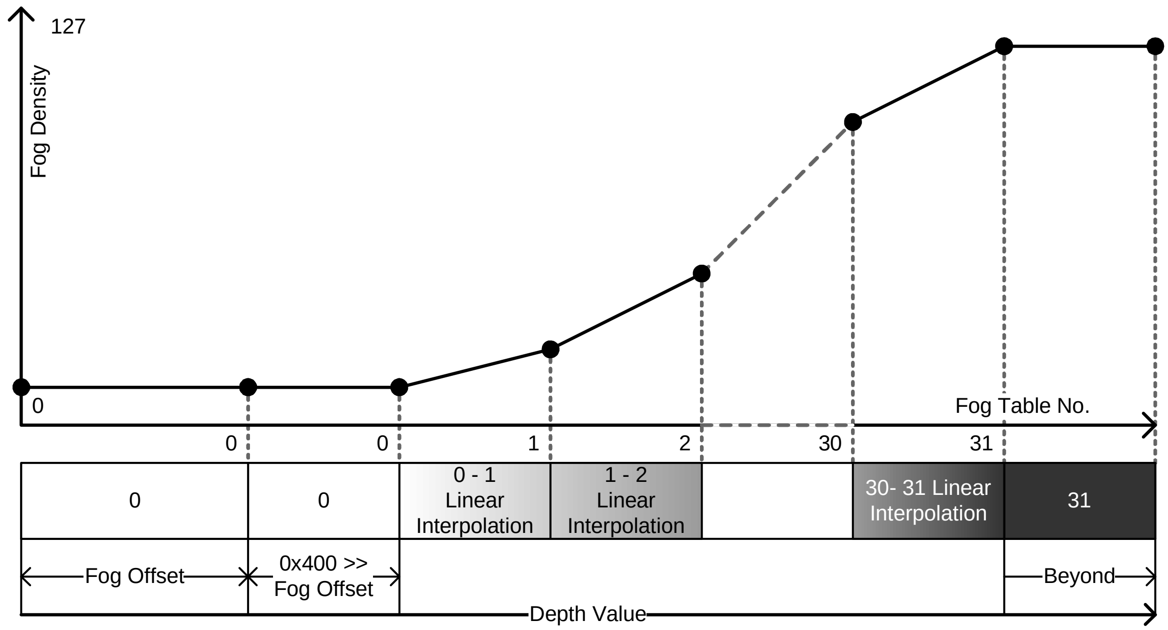
   Fog density f =

$$f = \frac{\{f(i) - f(i-1)\} \times \{Zd - (FOG\_OFFSET + F\_IVL \times i)\}}{F\_IVL} + f(i-1)$$

3. When FOG_OFFSET + F_IVL x 32 ≤ Zd ≤ 0x7FFF:

   Fog density f = f (31)

Figure 7-47: Depth Values and Fog Density



The fog shift is set by the DISP3DCNT register (the 3D Display Control register).

Table 7-19 shows the fog-blending equations.

Table 7-19: Fog-Blending Equations

Fog Mode	Fog Blending with Pixel's Color and α Value	Fog Blending only with Pixel's α Value
Fog Blending Equations	R = { f x Rf + (128 - f) x Rs} / 128 G = { f x Gf + (128 - f) x Gs } / 128 B = { f x Bf + (128 - f) x Bs } / 128 A = { f x Af + (128 - f) x As } / 128 Handling exceptions: When f = 127, (R, G, B, A) = (Rf, Gf, Bf, Af) is used	R = Rs G = Gs B = Bs A = { f x Af + (128 - f) x As } / 128 Handling exceptions: When f = 127, A = Af is used

(R, G, B, A) : Newly written fragment color (fractional parts resulting from calculations are truncated)

(Rf, Gf, Bf, Af) : Fog color

(Rs, Gs, Bs, As) : Color in the color buffer after edge marking

7.3.9.2 Fog Preprocessing for 2D

If fog is disabled, the region in which the color buffer's α value is zero-cleared is treated as an alpha cut-out region when compositing with the 2D screen. From this region, you can see the unmodified color of the 2D screen in the background. (See "7.4 2D Graphics Features You Can Apply to the 3D Screen After Rendering" on page 293.)

If, on the other hand, fog is enabled, this otherwise transparent region is also subject to fog blending, and the color buffer value is updated. Afterwards, the 2D color special-effect feature (2D and 3D alpha-blending) can work to alpha-blend the color buffer and the 2D screen, so fog is also applied to the 2D screen in the background seen from this region.

In this way, 2D fog blending is conducted via 2D and 3D alpha-blending. (See "7.3.7.2 2D and 3D Alpha-Blending Preprocess" on page 284.)

7.3.10 Anti-Aliasing

The anti-aliasing feature blends an edge section in the front buffer with the color in the back color buffer. The front color buffer holds the rendered results of the front polygon; the back buffer stores the rendered results of all polygons (including clear colors) that are behind the front polygon. Anti-aliasing is only applied to the edges of opaque polygons. Anti-aliasing uses the α value that is newly written into the color buffer as a blending factor when alpha-blending with 2D. Therefore, even a 2D background exhibits the anti-aliasing effect.

Table 7-20 shows the anti-aliasing equations.

Figure 7-48 illustrates the concept of anti-aliasing. Figure 7-49 shows the edge that is output to the LCD.

Table 7-20: Anti-Aliasing Equations

	Anti-aliasing Equations
A2 at least 1	RA = { (AC + 1) x R1 + (31 – AC) x R2 } / 32 GA = { (AC + 1) x G1 + (31 – AC) x G2 } / 32 BA = { (AC + 1) x B1 + (31 – AC) x B2 } / 32 AA = { (AC + 1) x 31 + (31 – AC) x A2 } / 32
A2 = 0	RA = R1 GA = G1 BA = B1 AA = { (AC + 1) x 31 } / 32

(RA, GA, BA, AA) : Color newly stored in the color buffer (fractional parts resulting from calculations are truncated).

(R1, G1, B1, 31) : Color in the front color buffer (alpha = 31 since the anti-aliasing target is an opaque polygon).

(R2, G2, B2, A2) : Color in the back color buffer

AC : Anti-aliasing factor (5 bits)

The anti-aliasing factor is applied proportionately to the surface area of the pixel occupied by the polygon.

Figure 7-48: Anti-Aliasing

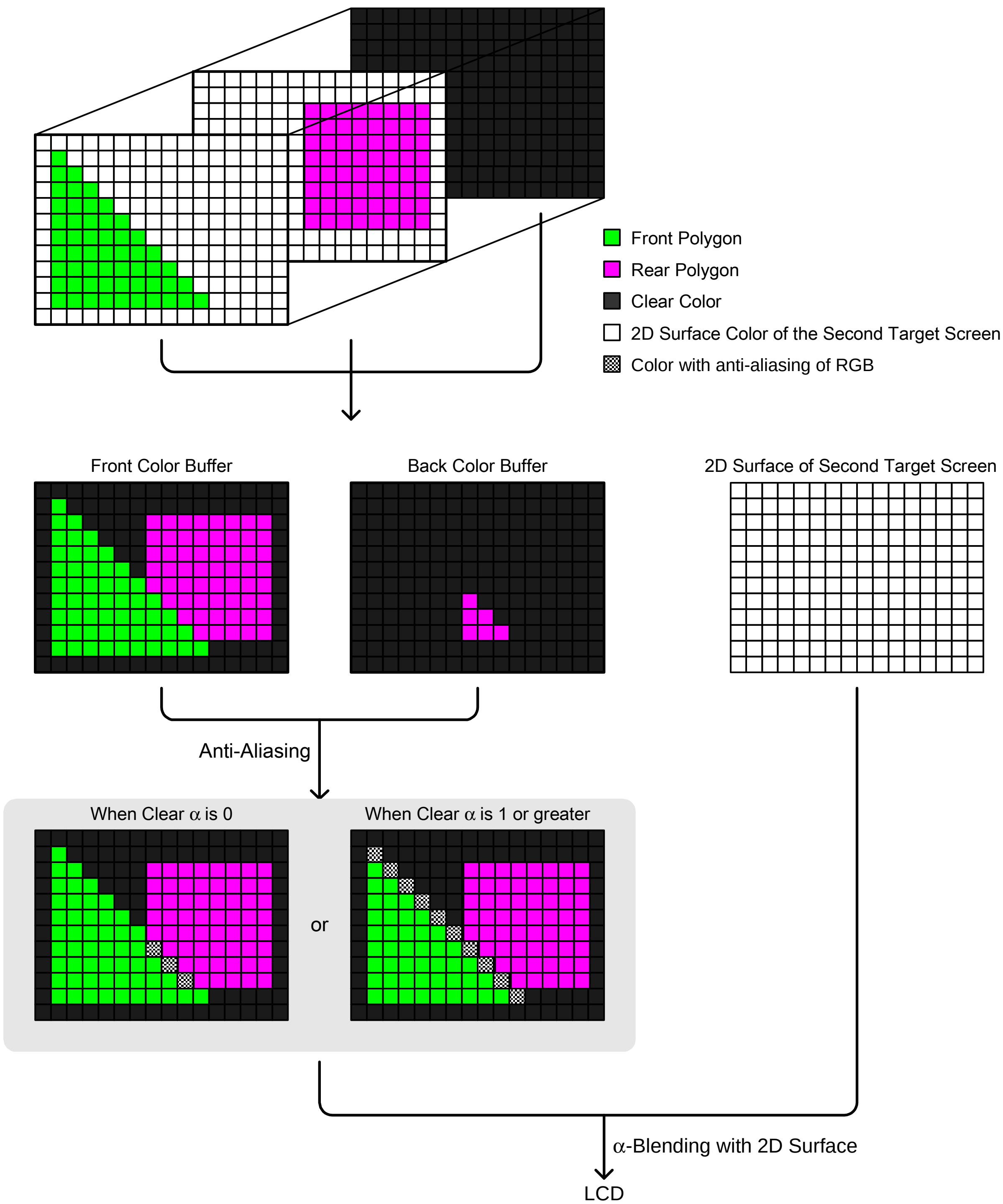


Figure 7-49: Final LCD Image Output (Anti-Aliasing)

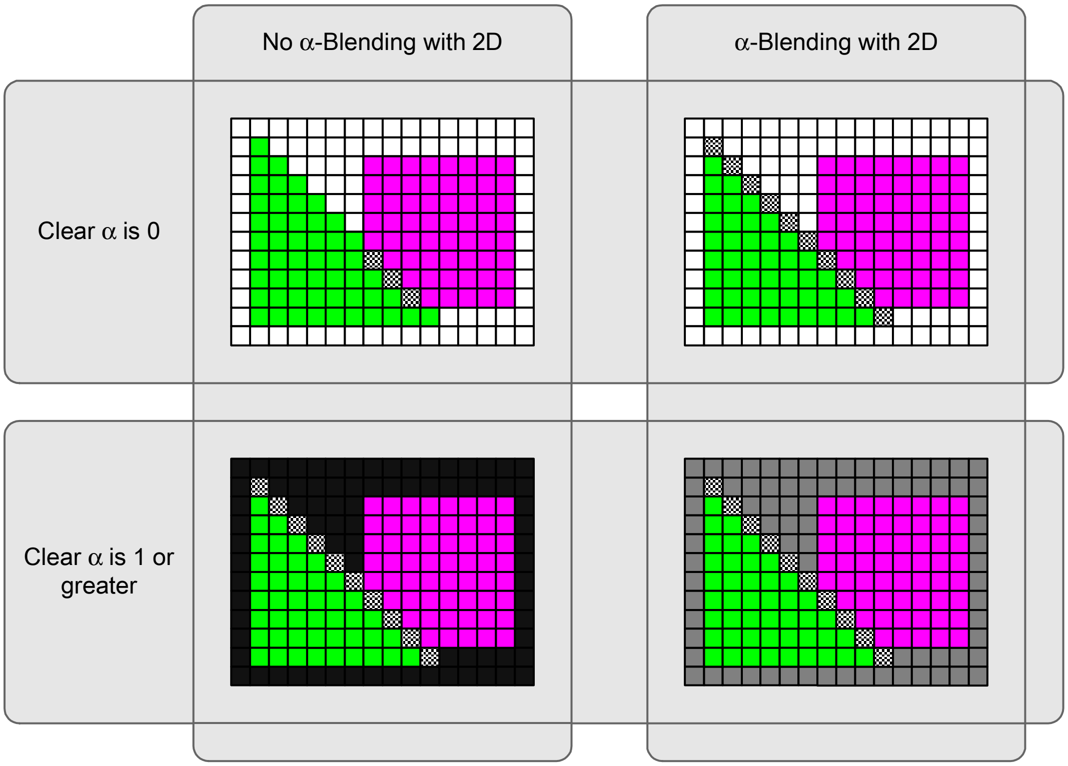


Figure 7-49 shows the visual image. Table 7-21 shows the actual RGB results

Table 7-21: Anti-Aliasing and Alpha-Blending with a 2D Surface

A2	α Blending with 2D Surface	Anti-aliasing Results (RGBA)		Result of α Blending with 2D Surface (RGB)	
		Edge	Background	Edge	Background
0	No	(R1, G1, B1, AA)	(R2, G2, B2, 0)	(R1, G1, B1)	Color of 2D surface
0	Yes	(R1, G1, B1, AA)	(R2, G2, B2, 0)	α-blended color of (R1, G1, B1) and 2D surface (α-blending factor AA)	Color of 2D surface
1 or More	No	(RA, GA, BA, AA)	(R2, G2, B2, A2)	(RA, GA, BA)	(R2, G2, B2)
1 or More	Yes	(RA, GA, BA, AA)	(R2, G2, B2, A2)	α-blended color of (RA, GA, BA) and 2D surface (α-blending factor AA)	α-blended result of (R2, G2, B2) and 2D surface (α-blending factor A2)

Because the interior of a polygon is opaque (alpha = 31), there is no alpha-blending with the 2D. Therefore, this case is omitted.

Table 7-21 shows how, when A2 is 0, anti-aliasing is applied only to alpha values. RGB color is written to the color buffer as is, with no blending.

In this case, do alpha-blending with the 2D surface and reflect this in RGB. (The alpha value from anti-aliasing is applied to alpha-blending.

7.3.11 Status

Rendered Line Count Register

Name: RDLINES_COUNT Address: 0x04000320 Attribute: R Initial value: 0x0000

15							8	7		5 0
										RENDERED_LINES_MIN
										Minimum value for number of rendered lines

- RENDERED_LINES_MIN[d05–d00] : Minimum value for number of rendered lines (0–46)

  Use this to check the minimum number of lines in the color buffer during display of the previous frame. This register is updated during every V cycle. The color buffer holds 48 lines, but 2 lines are the current buffer, so the largest count-value for this register is 46. You cannot confirm from this value whether or not lines have overflowed, but you can determine the risk of this happening. To determine whether or not lines have overflowed, check the Color Buffer Underflow Flag of the 3D Display Control register (DISP3DCNT).

  Note: When the counter reaches 0, there is a risk that the Rendering Engine will fail to draw lines (that is, lines will overflow). When the counter approaches 0, reduce the load on the Rendering Engine by, for example, reducing the number of polygons sent to the Geometry Engine.

<u>Comparison of Rendering Buffer Methods</u>

- **Normal frame-buffer method**

  In this method, there are two or three frame buffers for rendering and display, and the buffers are swapped during the V-Blank period immediately after drawing is completed. If there are more rendering polygons and pixels than the rendering engine can process, the frame rate drops due to rendering delays.

- **FIFO line-buffer method adopted by TWL**

  In this method, drawing and display involve the same FIFO buffer. This FIFO buffer has a capacity of 48 lines. During display, data is read from the FIFO buffer in sync with the timing of the LCD. Data for the horizontal direction is read with the dot clock, whereas data for the vertical direction is read in the horizontal scanning interval (355-dot clocks). If there are more polygons and pixels for the line to be drawn than the Rendering Engine can process, the display is corrupted because the Rendering Engine cannot render it in time.

7.4 2D Graphics Features You Can Apply to the 3D Screen After Rendering

In TWL, the 3D screen is displayed as BG0 after it has been rendered, rather than being displayed directly on the LCD. This enables certain 2D graphic features to also be applied and displayed on the LCD. To read the basic specifications for 2D graphics see "6 2D Graphics" on page 97.

7.4.1 Raster Scroll

Unlike 2D screens, 3D screens cannot be scrolled vertically. However, they can be scrolled horizontally.

BG0 Offset Settings Register

Name: BG0OFS   Address: 0x04000010   Attribute: W   Initial value: 0x0000

15–9	8	7–0
	SH	INTEGER_H
	H Offset	

Signed fixed-point number (sign + 8-bit integer)

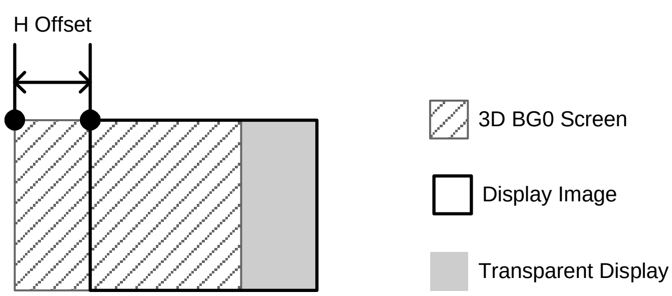
- H[d08–d00] : H offset

  Changes the starting position of display in the horizontal direction.

  Unlike for 2D screens, d08 is the sign bit, and the offset value can be set in the range of –256 to +256.

  Portions of the display screen that go beyond the screen because of horizontal scrolling become transparent. (See Figure 7-50.)

Figure 7-50: H Offset for a 3D Surface



7.4.2 Order of Display Priority with a 2D Screen

BG0 Control Register

Name: BG0CNT (x=0, 1)   Address: 0x04000008   Attribute: R / W   Initial value: 0x0000

15–2	1–0
	Order of Priority

By adjusting the display priority, you can place the 2D screen either in front of or behind the 3D screen.

See the diagram in "6.9 Display Priority" on page 175.

7.4.3 Windows

You can apply windows to BG0 of the 3D screen.

Window Position Settings Register

Name: WINxH (x=0, 1) Address: 0x04000040, 0x04000042 Attribute: W Initial value: 0x0000

15							8	7							0
Window's upper-left X coordinate								Window's lower-right X coordinate							

Name: WINxV (x=0, 1) Address: 0x04000044, 0x04000046 Attribute: W Initial value: 0x0000

15							8	7							0
Window's upper-left Y coordinate								Window's lower-right Y coordinate							

Window Inside Control Register

Name: WININ Address: 0x04000048 Attribute: R / W Initial value: 0x0000

15							8	7							0
		EFCT	OBJ	BG3	BG2	BG1	BG0			EFCT	OBJ	BG3	BG2	BG1	BG0
Window 1 Inside								Window 0 Inside							

Window Outside Control Register

Name: WINOUT Address: 0x0400004A Attribute: R / W Initial value: 0x0000

15							8	7							0
		EFCT	OBJ	BG3	BG2	BG1	BG0			EFCT	OBJ	BG3	BG2	BG1	BG0
OBJ Window Inside								Window (0, 1 and OBJ Window) Outside							

If the 3D screen has highest priority, α-blending is always enabled, regardless of the setting for the Window Control register's color-effect enable flag.

7.4.4 Color Effects

For details on each register parameter, see the Color Special Effects Register in 2D Graphics.

7.4.4.1 Alpha-Blending with the 2D Screen

The alpha-blending feature of 2D color effects is used for post-processing after the 2D and 3D alpha-blending process and the 2D fog process. To read about preprocessing, see "7.3.7.2 2D and 3D Alpha-Blending Preprocess" on page 284 and "7.3.9.2 Fog Preprocessing for 2D" on page 288.

Color Effect Control Register

Name: BLDCNT Address: 0x04000050 Attribute: R / W Initial value: 0x0000

15		13					8	7	6	5					0
		BD	OBJ	BG3	BG2	BG1	BG0	0	1	BD	OBJ	BG3	BG2	BG1	BG0
		Second Target Screen						Selected Effect		First Target Screen					

- [d07–d06] : Selected effect

  Perform alpha-blending by setting [d07] to 0 and [d06] to 1.

  The process involved when alpha-blending the 3D screen and the 2D screen differs, depending on the relative priority of the two screens. When the 2D screen is the first target screen, the value set in the BLDALPHA register is used for alpha-blending, as per the specifications. However, when the 3D screen is the first target screen, alpha-blending with the second target screen is performed using the alpha value that is being rendered to the color buffer (that is, alpha-blending is done in units of pixels).

Note: Any part with a color buffer alpha value of 0 is handled in the same way as a 2D cut-out region, and thus is not subjected to alpha-blending. Any part with a color buffer alpha value of 1 or more is subjected to alpha-blending.

Color Special Effect / Alpha-Blending Factors Register

Name: BLDALPHA Address: 0x04000052 Attribute: R / W Initial value: 0x0000

15			12				8	7			4				0
			EVB								EVA				

The factors used for the alpha-blending process are set by EVA and EVB in the BLDALPHA register. (EVA and EVB values that are 16 or above are treated as 16.)

7.4.4.2 Shininess Up/Down

Color Effect Control Register

Name: BLDCNT Address: 0x04000050 Attribute: R / W Initial value: 0x0000

15		13					8	7	6	5					0
		BD	OBJ	BG3	BG2	BG1	BG0	1		BD	OBJ	BG3	BG2	BG1	BG0
		Second Target Screen						Selected Effect		First Target Screen					

- [d07–d06] : Selection of color special effects

  Perform a process to change the shininess by setting [d07] to 1.

  When [d06] is set to 0, shininess is increased. When [d06] is set to 1, shininess is decreased.

  All zeros must be set for the second target screen.

Color Special Effect / Change Shininess Factor Register

Name: BLDY Address: 0x04000054 Attribute: W Initial value: 0x0000

15							8	7							0
											EVY				

The factor used for changing shininess is set by EVY in the BLDY register. (EVY values that are 16 or above are treated as 16.)

8 DMA

DMA is a high-speed data-transfer method that bypasses the CPU. It is controlled by the DMA Controller. The ARM9 bus on the TWL has four NITRO-compatible DMA channels (DMA0 – 3), and there are four new DMA channels (NDMA0-3) that are equipped with new features such as selecting an arbitration method between DMA, giving a total of eight DMA channels. (The ARM7 bus also has eight channels.)

The highest priority channel is DMA0, followed by DMA1, DMA2, and DMA3, in order of priority. For NDMA, the highest priority channel is NDMA0, followed by NDMA1, NDMA2, and NDMA3. However, when using the round-robin method for arbitration, ownership rights on the bus circulates around, so the relative priorities among NDMA0-3 no longer apply.

If a higher-priority DMA is activated while a lower-priority DMA is executing, the lower-priority DMA pauses, and the higher-priority DMA is executed. After the higher-priority DMA is finished, the lower-priority DMA resumes execution. Because DMA execution can be paused, consider giving higher priority to DMA transfers that must finish within a limited time frame.

However, when the CPU is operating with DMA, RAM outside the TCM or cache cannot be accessed.

Therefore, in the interval until the DMA finishes, an interrupt is delayed when processing anything other than TCM.

8.1 NITRO-Compatible DMA

The NITRO-compatible DMA is unchanged from NITRO except for circuit revisions for when multiple DMA channels are started in parallel.

DMAxSAD: DMAx Source Address Registers (x = 0 - 3)

Name	Address	Attribute	Initial Value
DMAxSAD (x =0 - 3)	0x040000B0, 0x040000BC, 0x040000C8, 0x040000D4	R/W	0x00000000

31–28	27–0 (31 · 27 · 24 | 23 · 16 | 15 · 8 | 7 · 0)
	DMA Source Address

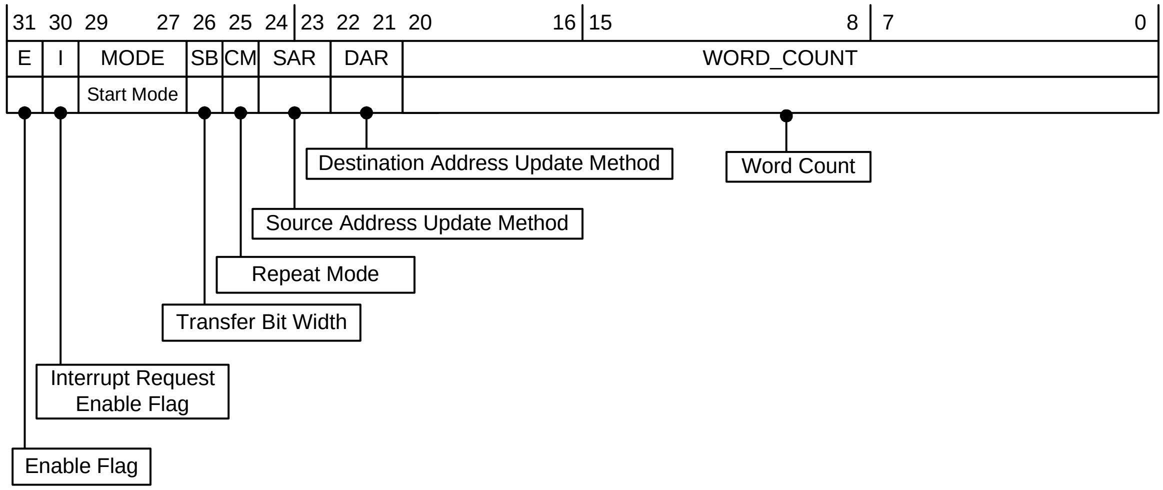
DMAxDAD: DMAx Destination Address Registers (x = 0 - 3)

Name	Address	Attribute	Initial Value
DMAxDAD (x =0 - 3)	0x040000B4, 0x040000C0, 0x040000CC ,0x040000D8	R/W	0x00000000

31–28	27–0 (31 · 27 · 24 | 23 · 16 | 15 · 8 | 7 · 0)
	DMA Destination Address

DMAxCNT: DMAx Control Registers (x=0 to 3)

Name	Address	Attribute	Initial Value
DMAxCNT (x =0 - 3)	0x040000B8,0x040000C4,0x040000D0,0x040000DC	R/W	0x00000000



- E[d31]: DMA enable flag

0	Disable
1	Enable

- I[d30]: Interrupt request enable flag

0	Disable
1	Enable

- MODE[d29–d27]: Options for DMA start mode

000	Start immediately
001	Start at V-Blank
010	Start at H-Blank (DMA does not start during an H-Blank within a V-Blank period)
011	Synchronize to start with display (that is, synchronized to start as each horizontal line is drawn)
100	Main memory display
101	TWL / DS Game Card
110	Game Pak
111	Geometry Command FIFO

- SB[d26]: Transfer bit-count selection

0	16 bits
1	32 bits

- CM[d25]: Repeat mode selected flag

0	Not Repeat mode
1	Repeat mode

- SAR[d24–d23]: Options to update source address

00	Increment
01	Decrement
10	Fixed
11	Setting prohibited

- DAR[d22–d21]: Options to update destination address

00	Increment
01	Decrement
10	Fixed
11	Increment/reload

- WORD_COUNT[d20–d00]: Word count
  Specifies the number of transfers.

- **Repeat mode**

  When the DMA repeat mode is on, DMA starts automatically each time the start mode conditions occur.

  If the repeat mode is not set, DMA stops when the transfer of the word count volume is complete.

  To cancel repeat mode, set the DMA enable flag to 0, as described in Step 2 in the procedure below.

- **Address update method**

  The details of processing the address update method are shown in Table 8-1.

Table 8-1 : Processing Details for the Address Update Method

Address Update Method	Process
Increment	The address value increases one unit with each transfer
Decrement	The address value decreases one unit with each transfer
Fixed	The address stays fixed
Increment/Reload	Increments for each transfer, and then returns to the transfer's starting address when the transfer of the word count is done

Note: Setting the address update method to fixed or decrement is prohibited if the source or destination is set to Game Pak space because the hardware does not support it.

- **DMA Start Mode**

  Main Memory Display Start Mode

  Do not set the DMA source address to any memory region outside of main memory in Main Memory Display Mode. Also, be sure to set the transfer bit mode to 32-bit and the word count value to 4.

  Geometry Command FIFO Start Mode

  When the Geometry Command FIFO is less than half full, DMA starts and 112 words (see the procedure below) are transferred. The process repeats until the volume transferred reaches the word count value.

  Note: If commands have been packed, the number of words sent in each repetition equals the number of words before unpacking.

- **Cautions When Starting and Stopping DMA**

  1. When starting DMA

     A delay of 2 cycles of the system clock (33.514 Mhz) occurs from the time the DMA enable flag is set until the time DMA starts. If any of the DMA-related registers are accessed during this period, DMA might not operate correctly. To prevent a DMA problem during this period, run another process, such as inserting a dummy Load command. (The main processor executes a Load command in ½ cycle[1] of the system clock, so you would need to insert two or more[2] Load commands to the same register.)

  2. When stopping DMA

     DMA begins when the signal that serves as the start trigger is issued. If the CPU disables DMA at the same time the start trigger is issued, DMA could lock up. Therefore, be sure to disable DMA at least 4 cycles after the start trigger.

     1. When the DMA repeat feature is off

        Because the DMA stops automatically after it is executed once, do not forcibly clear the DMA enable flag. Instead, wait for the flag to become 0.

     2. When the DMA repeat feature is on

        Be sure to clear the DMA enable flag with the CPU at least 4 cycles[3] after the signal that serves as the DMA start trigger. For example, you can safely stop DMA using the interrupt generated when DMA ends, clearing the DMA enable flag before the next start trigger is issued.

        If you cannot use this method, stop DMA by using the procedure described below.

     3. Stopping DMA in H-Blank or V-Blank auto-start mode

        During a V-Blank period, DMA is stopped, and the start trigger is not issued, so you can safely clear the DMA enable flag at that time. If you cannot use this method, follow the procedure below:

        Step 1: Write 16 bits to the DMA control register (see Table 8-2).

Table 8-2 : Register Configuration (Step 1)

Setting	Content
DMA Enable Flag	1 (Enable)

1. When operating at double speed in TWL mode, this will be 1/4 cycle.
2. When operating at double speed in TWL mode, this will be 4 or more.
3. When operating at double speed in TWL mode, this will be 8 cycles.

Setting	Content
DMA Start Timing	00 (Start Immediately mode)
DMA Repeat Mode	0 (Disable Repeat Mode)
Other Bits	Do not change

Step 2: Carry out the process for more than four cycles[1].

Example: (NITRO compatibility mode)

3 NOP or 1 LDR instruction) + 1st cycle of STR instruction from Step 3 = 4 cycles

The actual writing by the STR instruction occurs in the 2nd cycle.

Step 3: Write 16 bits to the DMA control register, then stop the DMA (see Table 8-3).

Table 8-3 : Register Configuration (Step 3)

Setting	Content
DMA Enable Flag	0 (Disable)
DMA Start Timing	00 (Start Immediately mode)
DMA Repeat Mode	0 (Disable Repeat Mode)
Other Bits	Do not change

Note: DMA may run one extra time in Step 1.

Note: In TWL mode, circuit revisions to the NITRO-compatible DMA controller can be enabled or disabled using the system configuration. If they are enabled, there is no need to consider the precautions shown below (the problems described will not be avoided in NITRO-compatible mode).

Precautions for starting multiple, parallel DMA channels in the ARM9 System Bus

When ARM946E-S starts accessing regions that cannot be accessed in a single system cycle (33.514 MHz), such as main memory, an ARM9-DMA with a lower priority (Auto) starts at the same time. The automatic startup of ARM9-DMA with a higher priority occurs immediately afterwards, and the DMA with higher priority runs out of control. This condition does not exist on ARM7 because the system bus specifications differ.

Workaround

Of the DMA Parallel Start Categories shown in Table 8-4, items in Category 3 must not be used together. In addition, start DMA from TCM. However, V-Blank start and H-Blank start can be used together.

1. When operating at double speed in TWL mode, this will be 8 cycles.

Table 8-4 : ARM9-DMA Parallel Start Category Chart

DMA Parallel Start Category Number	DMA Description
1	Start immediately
2	Geometry Command FIFO (Normal)
3	Geometry Command FIFO (Auto Start) V-Blank Start (can use with H-Blank Start) H-Blank Start (can use with V-Blank Start or with multiple H-Starts) Display Synchronization Main Memory Display Game Card

8.2 New DMA

The new DMA controller added to the TWL has the following features.

- It is possible to choose between the traditional fixed method or the round-robin method for DMA arbitration.
- It is possible to specify, for each channel, the number of words to transfer each time a single startup request is run, as well as the total number of words to transfer.
- It is possible to specify, for each channel, the number of words to transfer for single-block transfers, as well as the number of interval cycles between block transfers.
- The only transfer bit width is 32 bits.

NDMA Global Control Register

Name	Address	Attribute	Initial Value
NDMAGCNT	0x04004100	R/W	0x00000000

31	30–24	23–20	19–16	15–8	7–0
AM			CPUCYCLE		
			Cycle Selection		

- AM[d31] : DMA Arbitration Method

0	Fixed Method
1	Round-Robin (Revolving) Method

- CPUCYCLE[d19-d16] : Cycle Selection

  When round-robin is selected as the DMA arbitration method, this selects the number of cycles that can be executed when there is a request from the DSP or the ARM9 to the ARM9 bus. It will be ignored if there are no requests from the DSP or the ARM9 to the ARM9 bus.

0000	0 cycles	**1000**	128 cycles
0001	1 cycles	**1001**	256 cycles
0010	2 cycles	**1010**	512 cycles

0011	4 cycles	**1011**	1,024 cycles
0100	8 cycles	**1100**	2,048 cycles
0101	16 cycles	**1101**	4,096 cycles
0110	32 cycles	**1110**	8,192 cycles
0111	64 cycles	**1111**	16,384 cycles

NDMAx Source Address Registers (x = 0 - 3)

Name	Address	Attribute	Initial Value
NDMAxSAD (x =0 - 3)	0x04004104, 0x04004120, 0x04004158	R/W	0x00000000

31 27 24 23 16 15 8 7 0		
DMASRC		
DMA Source Address		

NDMAx Destination Address Registers (x = 0 - 3)

Name	Address	Attribute	Initial Value
NDMAxDAD (x =0 - 3)	0x04004108, 0x04004124, 0x04004140, 0x0400415C	R/W	0x00000000

31 24 23 16 15 8 7 2 0		
DMADEST		
DMA Destination Address		

- DMASRC, DMADEST [d31-d02] : Source (Destination) Address

  Zeroes are always output for the lower 2 bits of the DMA source address and the DMA destination address of the new DMA.

NDMAx Total Words Transferred Count Register (x = 0 - 3)

Name	Address	Attribute	Initial Value
NDMAxTCNT (x =0 - 3)	0x0400410C, 0x04004128, 0x04004144, 0x04004160	R/W	0x00000000

31 27 24 23 16 15 8 7 0				
				TOTALCNT
				Total Number of Words Transferred

- TOTALCNT[d27-d00] : Total Number of Words Transferred

  If the total number of words transferred is set to 0x00000000, the count will be set to 0x10000000. The total number of words transferred can be an integer multiple of the transferred word count or the word count of block transfers, and does not have to be a value higher than those counts.

NDMAx Word Count Register (x = 0 - 3)

Name	Address	Attribute	Initial Value
NDMAxWCNT (x =0 - 3)	0x04004110, 0x0400412C, 0x04004148, 0x04004164	R/W	0x00000000

31–24	23–16	15–8	7–0
	WORDCNT		
	Number of Words to Transfer		

- WORDCNT[d23-d00] : Number of Words to Transfer

  If the number of words to transfer is set to 0x000000, the count will be set to 0x1000000. The number of words to transfer can be an integer multiple of the number of words for block transfers, and does not have to be a value higher than those counts.

NDMAx Block Transfer Interval Register (x = 0 - 3)

Name	Address	Attribute	Initial Value
NDMAxBCNT (x =0 - 3)	0x04004114, 0x04004130, 0x0400414C, 0x04004168	R/W	0x00000000

31–18	17–16	15–0
	PS	ICNT
		Interval Timer

- PS[d17-d16] : Prescaler Selection

00	System Clock (33.514 MHz)
01	1/4th Frequency of the System Clock
10	1/16th Frequency of the System Clock
11	1/64th Frequency of the System Clock

- ICNT[d15-d00] : Interval Timer

  If the interval timer is set to 0x0000, the specified number of words to transfer will be transferred continuously until finished.

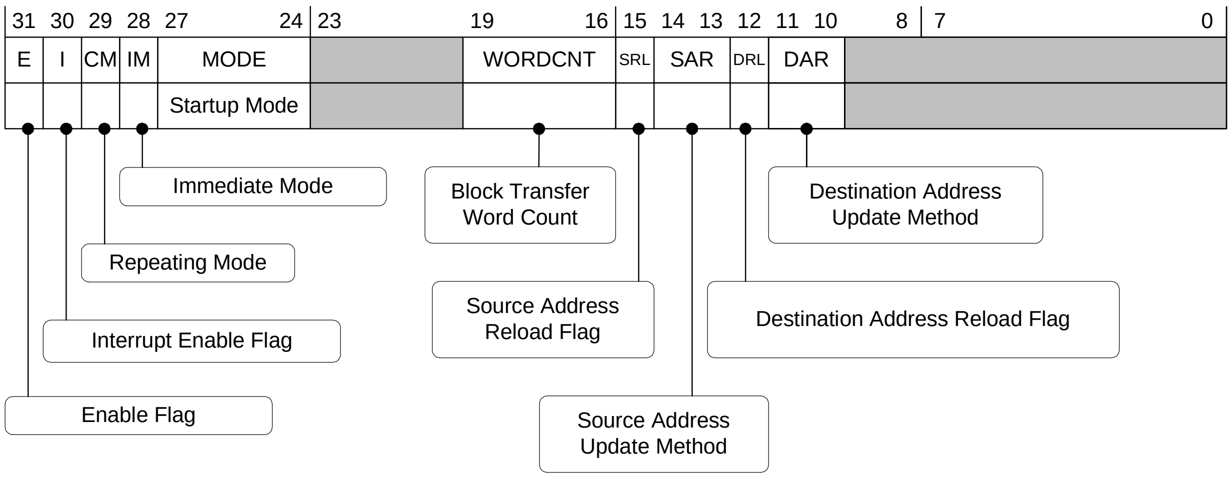
NDMAx Fill Data Register (x = 0 - 3)

Name	Address	Attribute	Initial Value
NDMAxFDATA (x =0 - 3)	0x04004118, 0x04004134, 0x04004150, 0x0400416C	R/W	0x00000000

31–0
FDATA
Fill Data

NDMAx Control Register (x = 0 - 3)

Name	Address	Attribute	Initial Value
NDMAxCNT (x =0 - 3)	0x0400411C, 0x04004138, 0x04004154, 0x04004170	R/W	0x00000000



- E[d31] : DMA Enable Flag

0	Disable
1	Enable

- I[d30] : Interrupt Enable Flag

0	Disable
1	Enable

- CM[d29] : Repeating Mode Selection Flag

0	Not in repeating mode
1	Repeating mode

- IM[d28] : Immediate Mode Selection Flag

0	Not in immediate mode
1	Immediate mode

- MODE[d27-d24] : DMA Startup Mode Selection

0000	Timer 0
0001	Timer 1
0010	Timer 2

0011	Timer 3
0100	TWL / NITRO Card
0101	Setting Prohibited
0110	Start up with V-Blank
0111	Start up with H-Blank (Will not start up with H-Blank during a V-Blank)
1000	Start up synchronized with the display (Start up synchronized with the start of H-Blank drawing during the display interval)
1001	Work RAM
1010	Geometry Command FIFO
1011	Camera
1100	Setting Prohibited
1101	Setting Prohibited
1110	Setting Prohibited
1111	Setting Prohibited

- WORCNT[d19-d16] : Block Transfer Word Count

0000	1 word	**1000**	256 words
0001	2 words	**1001**	512 words
0010	4 words	**1010**	1,024 words
0011	8 words	**1011**	2,048 words
0100	16 words	**1100**	4,096 words
0101	32 words	**1101**	8,192 words
0110	64 words	**1110**	16,384 words
0111	128 words	**1111**	32,768 words

- SRL[d15] : Source Address Reload Flag

0	Do not reload
1	Reload

- SAR[d15-d13] : Source Address Update Method Selection

00	Increment
01	Decrement
10	Fixed address
11	No address

- DRL[d12] : Destination Address Reload Flag

0	Do not reload
1	Reload

- DAR[d11-d10] : Destination Address Update Method Selection

00	Increment
01	Decrement
10	Fixed address
11	Setting prohibited

- Differences Between the Arbitration Methods

  There are two methods available for DMA arbitration: the fixed method, which has the fixed priorities NDMA0 > NDMA1 > NDMA2 > NDMA3, and the round-robin method, which rotates the bus ownership in the direction of NDMA0 to NDMA3. These methods can be set using the Global Control Register (NDMAGCNT).

  With the fixed method, if a request arrives whose priority is higher than the currently running DMA transfer, the current transfer will be paused once the specified number of words to transfer have completed their block transfer, and the higher-priority DMA transfer will begin. Conversely, if a request arrives whose priority is lower than the currently running DMA transfer, the incoming request will be put on hold until the current DMA transfer is completed.

  With the round-robin method, the system will look up the requested DMA channels, and once the specified number of words to transfer specified for each channel have completed their block transfer, bus ownership will be transferred to the next entity (in the order NDMA0, NDMA1, NDMA2, NDMA3, DSP, and ARM9), including the DMA channel that is running, after which the DMA transfer for the channels and the bus access for the DSP and ARM9 will begin.

  In addition, the round-robin method makes it possible to specify the number of accessible cycles for the DSP or ARM9 bus. This can be specified using the Cycle Selection bits of the Global Control Register.

- Immediate Mode and Repeating Mode

  Immediate Mode will start DMA with immediate mode selected and Enable Flag set to enabled. DMA transfers started in Immediate Mode will set the Enable Flag to disabled once the number of words to transfer specified in the Words Transferred Counter Register (NDMAxWCNT) have completed. When started in Immediate Mode, the Repeating Mode Selection Flag and the Total Number of Words Transferred setting will be ignored.

  Repeating Mode will start DMA with repeating mode selected, set the Enable Flag to enabled, and then start the DMA request for the hardware selected in the DMA Startup Mode Selection bits. DMA Transfers started in Repeating Mode will repeatedly transfer their number of words to transfer until the Enable Flag is set to disabled. When started in Repeating Mode, the TOTALCNT setting is ignored.

  DMA transfers that occur neither in Immediate Mode nor Repeating Mode will transfer the number of words to transfer until the total number of words to transfer is reached. After completion, the Enable Flag is set to disabled.

- Block Transfers

  With the new DMA, once DMA transfers begin, the bus will be monopolized until the selected number of words for block transfers has completed. A single block transfer cycle will never be split up.

  Once the first block transfer completes, the next block transfer will start after the number of cycles specified in the interval timer. The interval timer is a down counter, so it will count down for each selected prescaler, and once it reaches zero, the DMA request for the next block transfer will be issued. If the interval timer is set to 0, the specified number of words to transfer will be transferred continuously until finished.

- Reloading Addresses

  If the source address and destination address are set to reload, after the number of words to transfer have been completely transferred, the incremented or decremented address will be reset to the address used when the transfer was started.

- The Fill Data Register

  By selecting "No address" as the update method for the source address, it is possible to skip the read cycle and only write the value set for the Fill Data Register (NDMAxFDATA) during the write cycle to the destination address. Since this occurs only during the write cycle, this lets you clear a region to a fixed value faster than is possible with a standard DMA transfer.

9 Timer

The ARM9 bus side of TWL comes equipped with a 4-channel, 16-bit timer.

When the timer is enabled, the Count Register counts up according to the *prescaler* (frequency divider) cycle specified with the Control Register.

An interrupt can be generated when the Count Register overflows.

If the Count Register overflows, the value set when the count began is loaded, and the count starts over.

TM0CNT_L: Timer 0 Count Register

Name: TM0CNT_L Address: 0x04000100 Attribute: R/W Initial Value: 0x0000

15							8	7							0
Timer 0 Counter															

- [d15–d00]: Timer 0 Counter

TM0CNT_H: Timer 0 Control Register

Name: TM0CNT_H Address: 0x04000102 Attribute: R/W Initial Value: 0x0000

15							8	7	6					1	0
								E	I					PS	
														Prescaler	

- [d07–d00]: Timer 0 Control
  - E[d07]: Timer 0 Enable Flag

0	Disable
1	Enable

  - I[d06]: Interrupt Request Enable Flag

0	Disable
1	Enable

  - PS[d01–d00]: Prescaler Selection Flag

00	System Clock (33.514 Mhz)
01	1/64 of System Clock
10	1/256 of System Clock
11	1/1024 of System Clock

TMxCNT_L: Timer x Count Register (x = 1 – 3)

Name: TMxCNT_L (x =1–3) Address: 0x04000104,0x04000108,0x0400010C Attribute: R/W Initial Value: 0x0000

15							8	7							0
Timer x Counter															

- [d15–d00]: Timer x Counter

TMxCNT_H: Timer x Control Register (x = 1 – 3)

Name: TMxCNT_H (x =1–3) Address: 0x04000106,0x0400010A,0x0400010E Attribute: R/W Initial Value: 0x0000

15							8	7	6				2	1	0
								E	I				CH	PS	
													Multi-Stage	Prescaler	

- [d07–d00]: Timer x Control
  - E[d07]: Timer x Enable Flag

0	Disable
1	Enable

  - I[d06]: Interrupt Request Enable Flag

0	Disable
1	Enable

  - CH[d02]: Multistage Counter Selection Flag

0	According to Prescaler setting
1	Counts up when timer (x-1) overflows regardless of Prescaler setting

  - PS[d01–d00]: Prescaler Selection Flag

00	System Clock (33.514Mhz)
01	1/64 of System Clock
10	1/256 of System Clock
11	1/1024 of System Clock

10 Interrupts

This chapter describes the hardware interrupts for the ARM9 main processor.

When an interrupt request signal occurs from each piece of hardware, the bit that supports the interrupt request register is set, and if interrupts are enabled, the CPU is informed of the interrupt occurrence.

Each hardware interrupt request signal can be disabled individually via the interrupt enable register.

10.1 Interrupt Master Enable Register

This register can disable registers as a whole, and it configures whether to disable all registers or to enable the interrupt enable register settings.

IME: Interrupt Master Enable Register

Name: IME Address: 0x04000208 Attribute: R/W Initial Value: 0x0000

15							8	7							0
															IME

- IME[d00]: Interrupt Master Enable Flag

0	Disable all interrupts
1	Enable the Interrupt Enable Register settings

10.2 Interrupt Enable Register

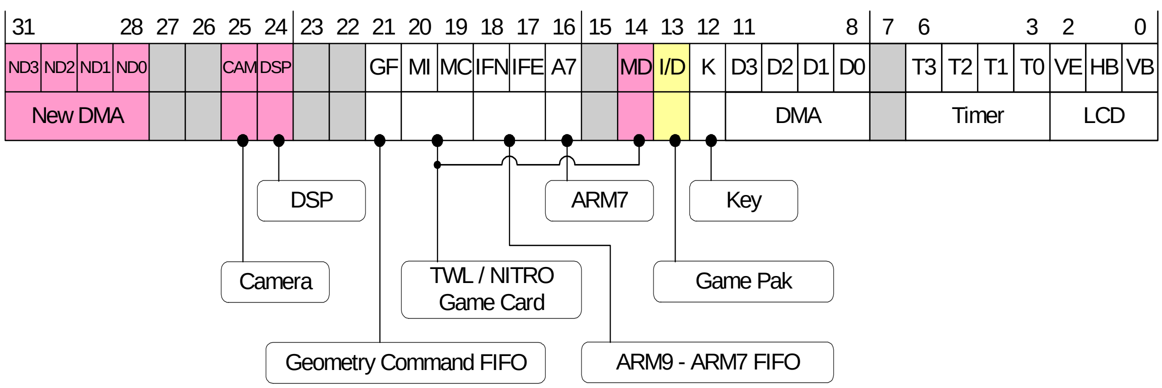
Each hardware interrupt request can be disabled individually.

Setting each bit enables interrupt requests from the corresponding hardware. Conversely, interrupt requests from corresponding hardware are disabled when the bit is reset.

The pink areas in the register table are the bits that were added with TWL. If the extended features are turned off in the system configuration in TWL mode, they will all be fixed at 0. The yellow area in the register table is the bit whose specifications has changed on the TWL.

IE: Interrupt Enable Register

Name: IE Address: 0x04000210 Attribute: R/W Initial Value: 0x00000000



- ND3[d31]: New DMA3 Interrupt Permission Flag
- ND2[d30]: New DMA2 Interrupt Permission Flag
- ND1[d29]: New DMA1 Interrupt Permission Flag
- ND0[d28]: New DMA0 Interrupt Permission Flag

  For more information, see "8 DMA" on page 297.
- CAM[d25]: Camera Interrupt Permission Flag
- DSP[d24]: DSP Interrupt Permission Flag
- GF[d21]: Geometry Command FIFO Interrupt Permission Flag

  For more information, see "7.2.16 Status" on page 245.
- MI[d20]: TWL/NITRO Card IREQ_MC Interrupt Permission Flag
- MC[d19]: TWL/NITRO Card Data Transfer Completion Interrupt Permission Flag
- IFN[d18]: ARM9 – ARM7 FIFO Not Empty Interrupt Permission Flag
- IFE[d17]: ARM9 – ARM7 FIFO Empty Interrupt Permission Flag
- A7[d16]: ARM7 Interrupt Permission Flag
- MD[d14]: TWL/NITRO Card MC_DET Interrupt Permission Flag
- I/D[d13]: Game Pak IREQ/DREQ Interrupt Permission Flag

  Because the Game Pak functionality has been removed, this interrupt will not occur.
- K[d12]: Key Interrupt Permission Flag

For more information, see "13.2 Interrupt Handling for Key Input" on page 330.

- D3[d11]: DMA3 Interrupt Permission Flag
- D2[d10]: DMA2 Interrupt Permission Flag
- D1[d09]: DMA1 Interrupt Permission Flag
- D0[d08]: DMA0 Interrupt Permission Flag

For more information, see "8 DMA" on page 297.

- T3[d06]: Timer 3 Interrupt Permission Flag
- T2[d05]: Timer 2 Interrupt Permission Flag
- T1[d04]: Timer 1 Interrupt Permission Flag
- T0[d03]: Timer 0 Interrupt Permission Flag

For more information, see "9 Timer" on page 309.

- VE[d02]: V-Counter Match Interrupt Permission Flag
- HB[d01]: H-Blank Interrupt Permission Flag
- VB[d00]: V-Blank Interrupt Permission Flag

For more information, see "5.3 Display Status" on page 75.

10.3 Interrupt Request Register

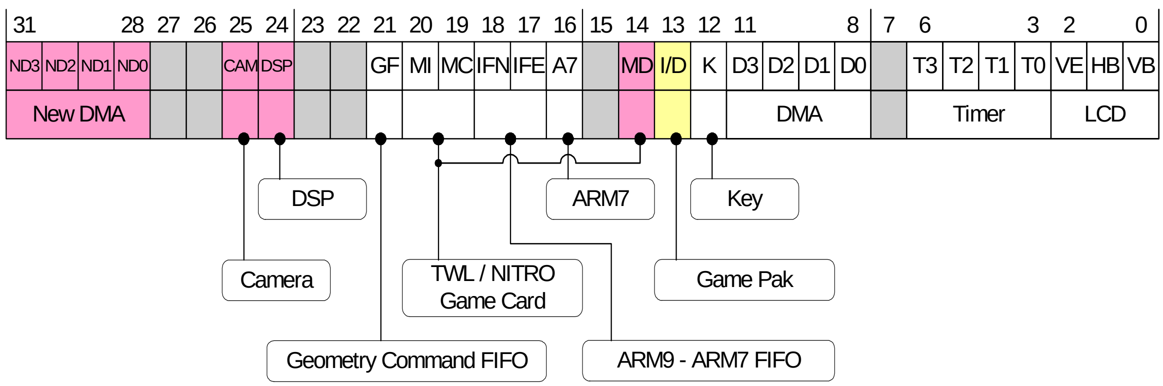
When an interrupt request from a hardware component occurs, the corresponding bit for the hardware component is set in the interrupt request register.

Also, if 1 is written to the bit where the interrupt request flag is set, the interrupt request flag is reset.

The pink areas in the register table are the bits that were added with TWL. If the extended features are turned off in the system configuration in TWL mode, they will all be fixed at 0. The yellow area in the register table is the bit whose specifications has changed on the TWL.

IF: Interrupt Request Register

Name: IF Address: 0x04000214 Attribute: R/W Initial Value: 0x00000000



- ND3[d31]: New DMA3 Interrupt Request Flag
- ND2[d30]: New DMA2 Interrupt Request Flag
- ND1[d29]: New DMA1 Interrupt Request Flag
- ND0[d28]: New DMA0 Interrupt Request Flag

  For more information, see "8 DMA" on page 297.
- CAM[d25]: Camera Interrupt Request Flag
- DSP[d24]: DSP Interrupt Request Flag
- GF[d21]: Geometry Command FIFO Interrupt Request Flag

  For more information, see "7.2.16 Status" on page 245.
- MI[d20]: TWL/NITRO Card IREQ_MC Interrupt Request Flag
- MC[d19]: TWL/NITRO Card Data Transfer Completion Interrupt Request Flag
- IFN[d18]: ARM9 – ARM7 FIFO Not Empty Interrupt Request Flag
- IFE[d17]: ARM9 – ARM7 FIFO Empty Interrupt Request Flag
- A7[d16]: ARM7 Interrupt Request Flag
- MD[d14]: TWL/NITRO Card MC_DET Interrupt Request Flag
- I/D[d13]: Game Pak IREQ/DREQ Interrupt Request Flag

  Because the Game Pak functionality has been removed, this interrupt will not occur.
- K[d12]: Key Interrupt Request Flag

For more information, see "13.2 Interrupt Handling for Key Input" on page 330.

- D3[d11]: DMA3 Interrupt Request Flag
- D2[d10]: DMA2 Interrupt Request Flag
- D1[d09]: DMA1 Interrupt Request Flag
- D0[d08]: DMA0 Interrupt Request Flag

For more information, see "8 DMA" on page 297.

- T3[d06]: Timer 3 Interrupt Request Flag
- T2[d05]: Timer 2 Interrupt Request Flag
- T1[d04]: Timer 1 Interrupt Request Flag
- T0[d03]: Timer 0 Interrupt Request Flag

For more information, see "9 Timer" on page 309.

- VE[d02]: V-Counter Match Interrupt Request Flag
- HB[d01]: H-Blank Interrupt Request Flag
- VB[d00]: V-Blank Interrupt Request Flag

For more information, see "5.3 Display Status" on page 75.

10.4 Interrupt Cautions

10.4.1 Clearing IME and IE

Even while the command to clear all flags in the IME and IE registers is executing, relevant interrupts are generated.

When clearing the IE flags, be sure to clear IME first to avoid inconsistencies in interrupt checks.

10.4.2 Multiple Interrupts

If clearing of IME and an interrupt occur at the same time, multiple interrupts are not accepted during that interrupt. Therefore, you must set IME after clearing it during the interrupt routine.

10.4.3 Interrupt Delays During DMA Operation

The CPU cannot access RAM other than the TCM or cache RAM during DMA operations.

Therefore, during the interval until the DMA stops, the interrupt is delayed when performing interrupt handling on anything other than TCM.

10.4.4 Interrupts from ARM7

The A7, IFE, and IFN interrupts are for use by the subprocessor and the subprocessor API for communications.

The subprocessor API will not operate properly if these interrupts are disabled or if the interrupt request is reset.

10.4.5 Interrupts from the DSP

DSP interrupts are for use by the DSP API for communications with the DSP.

The DSP API will not operate properly if these interrupts are disabled or if the interrupt request is reset.

10.4.6 Interrupts from the Camera

CAM interrupts are for use by the Camera API for communications with the camera.

The Camera API will not operate properly if these interrupts are disabled or if the interrupt request is reset.

11 Power Management

As with NITRO, the Power Management API of the TWL can be used by the application to put the TWL into Sleep mode, to control power to various circuits, to check for the low-battery state, and to check whether the system is open or closed.

11.1 Sleep Mode

The application can use the Power Management API to put the system into Sleep mode. In Sleep mode, all circuits in the TWL Processor stop. Power to the LCD and the sound is turned off, so there is nothing displayed and no sounds are played. However, the data in the TWL Processor internal memory and in main memory are retained.

The Power LED blinks slowly in Sleep mode. A fully charged battery lasts about ten days in Sleep mode.

Table 11-1 shows the factors that cause the system to recover from Sleep mode and the timing with which this happens.

Table 11-1 : Conditions for Recovering from Sleep Mode

Conditions for Waking	Timing
TWL System Is Opened	When TWL is opened
RTC Alarm Feature	When the alarm reaches the set time
TWL/NITRO Game Card	When a TWL/NITRO Game Card is accidentally removed
Key Entry	When a previously specified key (with the exception of X or Y) is pressed

Note: Because the Game Pak slot has been removed, and interrupts from the Game Paks are constantly occurring, it is not possible to transition to Sleep mode when Game Pak interrupts are permitted.

Note: In Sleep mode, it is difficult to perform shutdown processing in the same way as you would in Active mode since the overall system status is different from that of Active mode. For this reason, we perform any necessary processing in advance when making the transition to Sleep mode, and do not perform any shutdown processing while in Sleep mode. With TWL, the shutdown process that was registered using the Power Management API will be called automatically when the system transitions to Sleep mode. Consequently, even if the power is cut off suddenly during Sleep mode, the system will behave as though the power had been turned off in Active mode.

11.2 Controlling Various Power Supplies

The Power Management API can be used to control the power supply to the sound circuitry, LCD backlight, LCD, microphone, system, and graphics.

11.2.1 Sound

It is possible to control the power supply to the sound circuitry. However, that function is not currently being disclosed.

11.2.2 LCD Backlight

The Power Management API can be used to control the power supply to the backlight of the upper screen and the backlight of the lower screen separately. When the application is using only one LCD screen, the power can be turned off to the screen that is not being used to reduce battery consumption.

When no game is being played and the system is not standing by for wireless, if you choose not to display on either LCD and to turn off power to the LCD backlight in consideration of battery life, we suggest that you move to Sleep mode, which more effectively reduces power consumption.

11.2.3 LCD

The Power Management API can be used to control power supply to both the upper and lower screen LCDs. Furthermore, the LCD backlight can be turned off regardless of the LCD backlight settings. However, the LCD backlight settings are preserved.

When there is no display on the LCD screen, such as when the application is waiting for wireless communication, battery consumption can be controlled by turning off the power supply to the LCD.

When the system is not standing by for wireless, if you choose to turn off power to the LCD in consideration of battery life, we suggest that you move to Sleep mode, which more effectively reduces power consumption.

Note: Although the power supply to the LCD can be directly controlled with the graphics power control register mentioned below, the LCD circuitry may be damaged according to when the LCD is turned on or off. Therefore, directly changing the register value is prohibited. When manipulating the power supply to the LCD, always use the API.

When the LCD is turned off, the power supply for the sound amp is also turned off, and the speaker will not function. However, if headphones are connected when the LCD is on, and then the LCD is turned off, sound can still be played through the headphones. Also, unlike NITRO, the TWL guarantees that sound will be played through the headphones, even if the headphones are connected when the LCD is off.

11.2.4 Microphone

When using the microphone, power must be provided to the programmable gain amp (PGA). The Power Management API can be used to control the provision of power to the PGA.

Note: Do not use the microphone for 3 seconds after it is turned on.

11.2.5 System

The Power Management API can be used to perform a system reset or turn off the TWL's power (shutdown).

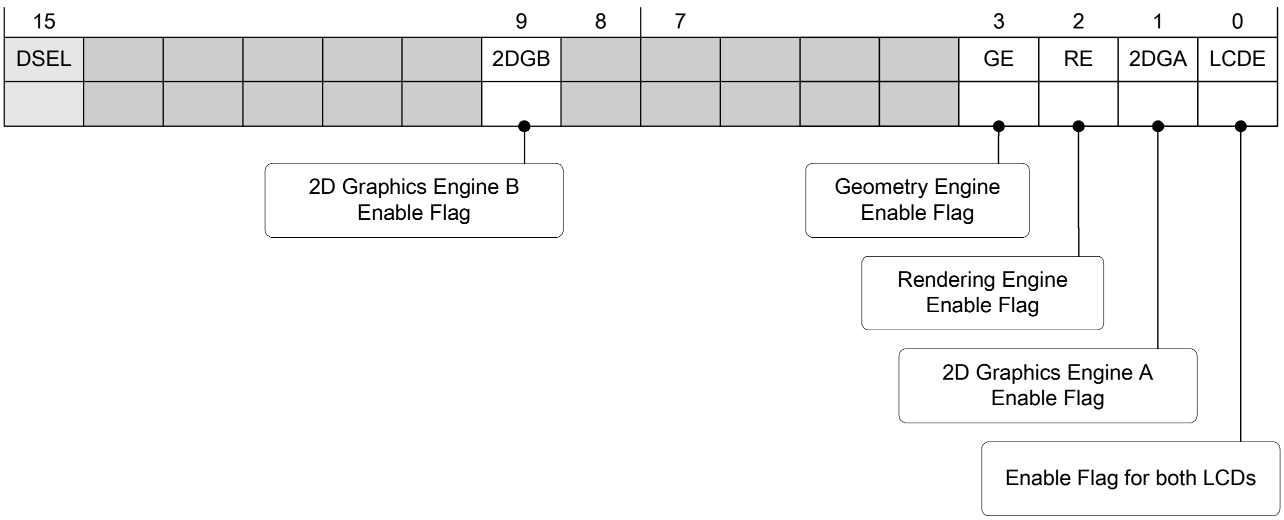
Note: Depending on the unit, there is no guarantee that the system will be reliably shut down if a shutdown is executed when the LCD is off (in rare instances the system may restart). Therefore, be sure to perform a shutdown only when the LCD is on.

11.2.6 Graphics

Power consumption can be reduced by: controlling the clock supply to the circuits of the geometry engine, the rendering engine, and the 2D graphics engine; and disabling circuits that are not being used.

Figure 11-1 : POWCNT: Graphics Power Control Register

Name: POWCNT Address: 0x04000304 Attribute: R/W Initial value: 0x0000



- 2DGB[d09] : 2D Graphics Engine B Enable Flag

  Used to reduce power consumption when the 2D Graphics Engine B is not being used.

0	Disable
1	Enable

- GE[d03] : Geometry Engine Enable Flag

  When the geometry engine is enabled, issue the SwapBuffers command once.

0	Disable
1	Enable

- RE[d02] : Rendering Engine Enable Flag

  If the rendering engine is enabled, issue the SwapBuffers command once.

0	Disable
1	Enable

- 2DGA[d01] : 2D Graphics Engine A Enable Flag

  Used to reduce power consumption when only 3D graphics are being used.

0	Disable
1	Enable

- LCDE[d00] : Enable Flag for Both LCDs (Use Prohibited)

  When disabled, both the clock supply to the LCD main and sub-controllers and the power supply to the main and sub-LCDs are stopped.

0	Disable
1	Enable

Note: Use the API to enable/disable the LCDs. Be careful not to change this bit when writing data to other bits.

The memory addresses and registers to which the clock signal is stopped when each flag is disabled:

- When 2D Graphics Engine A is disabled:

  0x04000008 - 0x0400004D

  0x04000050 - 0x04000055

  2D Graphics Engine A's OAM and palette RAM

- When 2D Graphics Engine B is disabled:

  0x04001008 - 0x0400104D

  0x04001050 - 0x04001055

  2D Graphics Engine B's OAM and palette RAM

- When the Geometry Engine is disabled:

  0x04000400 - 0x04000473

  0x04000480 - 0x040004AF

  0x040004C0 - 0x040004D3

  0x04000500 - 0x04000507

  0x04000540 - 0x04000543

  0x04000580 - 0x04000583

  0x040005C0 - 0x040005CB

  0x04000600 - 0x04000607

  0x04000610 - 0x04000611

  0x04000620 - 0x04000635

  0x04000640 - 0x040006A3

- When the Rendering Engine is disabled:

  0x04000320 - 0x04000321

  0x04000330 - 0x04000341

  0x04000350 - 0x0400035D

  0x04000360 - 0x040003BF

Table 11-2 shows the behavior that occurs when there is access to memory and registers to which the clock signal has been stopped.

Table 11-2 : Access to Memory and Registers when Clock Signal is Stopped

	Write	Read
Memory	Invalid	ALL zero
Registers	Invalid	Read-enabled

11.3 Power Status

11.3.1 Low Battery State and Battery Level

When the battery level drops below 10-20%, the low battery state is entered, and the power indicator LED turns red. When the battery level drops to about 1%, the power indicator LED begins a slow, red blinking pattern. Applications have no control over this behavior. The Power Management API can be used to read the battery state data and check directly whether the system is in the low-battery state and what the battery level is in one of 6 levels (see Table 11-3). The amount of charge left in the battery when the power indicator LED is solid red or in the slow, red blinking pattern is only a rough indication due to individual differences in the system, batteries, application, and environmental temperature.

Table 11-3 : Battery State Data

Data Type	Data Content
Low Battery State	Low battery state flag (0 – 1)
Battery Level	Battery level (0 – 5)

Details about PMIC status data:

- Low Battery State Flag

  0 : Battery still has a charge.

  1 : Battery is low.

- Battery Level

  0 : 0% battery level (Empty)

  1 : 0-1% battery level (Low)

  2 : 1-10% battery level

  3 : 10-30% battery level

  4 : 30-60% battery level

  5 : 60-100% battery level

Callback functions can be called when the battery level changes from 2 to 1, or from 1 to 0. These callbacks can be used to detect drops in the battery level without having to poll it.

Note: Applications that access NAND Flash memory or SD Memory Cards will shut down automatically when there is a little extra remaining battery charge. This is done to protect the file management regions in NAND Flash memory and SD Memory Cards. As a result, such applications will not be able to completely drain the battery. Applications that only access the Game Card or the backup memory on the Game Card will not shut down the system automatically when the battery level runs low. Such applications can be played until the battery is completely drained, as with NITRO,

but they are unable to detect when the battery level reaches 0, so the corresponding callback will not be called. Furthermore, if the backlight is suddenly changed to a higher brightness setting when the battery level is at 2, the power may cut off suddenly.

11.3.2 TWL Open/Closed State

The Power Management API can be used to read the status data shown in Table 11-4 to check whether the TWL is open or closed.

Table 11-4 : TWL Opened/Closed State Data

Data Type	Data Content
Device Opened/Closed State	TWL Opened/Closed State Flag (0 – 1)

Details about the TWL open/closed status data:

- TWL Opened/Closed State Flag

  0 : TWL is open.

  1 : TWL is closed.

12 Accelerators

TWL contains both divider and square root computation accelerators. These are the same as those found in NITRO except for the revisions to the divider circuit.

12.1 Divider

Shown below are the Divider Data setting register for the Divider and the Control Register that specifies the Divider mode and indicates the Divider state.

Divider Data (Numerator, Denominator, Quotient, Remainder) Registers

Name	Address	Attribute	Initial Value	Comment
DIV_NUMER	0x04000290	R/W	0x00000000_00000000	Numerator
DIV_DENOM	0x04000298	R/W	0x00000000_00000000	Denominator
DIV_RESULT	0x040002A0	R/W	0x00000000_00000000	Quotient
DIVREM_RESULT	0x040002A8	R/W	0x00000000_00000000	Remainder

31 … 24	23 … 16	15 … 8	7 … 0
Divider Data Lower Word			

31 … 24	23 … 16	15 … 8	7 … 0
Divider Data Upper Word			

Signed integer (sign + 63-bit integer part)

DIVCNT: Divider Control Register

Name: DIVCNT Address: 0x04000280 Attribute: R/W Initial value: 0x0000

15	14	13 … 2	1 … 0
BUSY	DIV0		MODE
Busy	Division by Zero		Division Mode

- BUSY[d15]: Busy Flag

0	Divider is ready
1	Divider is busy

- DIV0[d14]: Divide-by-zero error flag

0	There is no divide-by-zero error
1	There is a divide-by-zero error

TWL-06-0017-001-D
Released: February 16, 2009

- MODE[d01–d00]: Divider Mode

00	32-bit (DIV_NUMER)/32-bit (DIV_DENOM) quotient 32-bit (DIV_RESULT), remainder 32-bit (DIVREM_RESULT)
01	64-bit (DIV_NUMER)/32-bit (DIV_DENOM) quotient 64-bit (DIV_RESULT), remainder 32-bit (DIVREM_RESULT)
10	64-bit (DIV_NUMER)/64-bit (DIV_DENOM) quotient 64-bit (DIV_RESULT), remainder 64-bit (DIVREM_RESULT)
11	Setting prohibited

In TWL mode, it is possible to enable or disable the circuit revisions for the divider using the SCFG_EXT register in system configuration. When enabled, there is no need to consider the precautions shown below (these problems will not be resolved in NITRO-compatible mode).

Note: Regardless of the Divider mode, the Division by Zero Error flag is enabled only when all 64 bits of the denominator (DIV_DENOM) are zero.

For this reason, set all of the upper 32 bits of the denominator (DIV_DENOM) to 0 even when the Divider mode is 32-bit/32-bit or 64-bit/32-bit.

If the upper 32 bits of the denominator (DIV_DENOM) are not set to 0, the Division by Zero Error flag will not function properly.

12.1.1 Number of Calculation Cycles

After writing to the Divider Data Registers, the DIVCNT register's busy flag is set during the cycles shown in Table 12-1, according to the Divider Mode. When the busy flag has been cleared, you can find the calculation result by reading the register that stores the result.

Table 12-1 : Calculation Bit Count and Calculation Cycle Count by Divider Mode

Divider Mode	Calculation Bit Count	Calculation Cycle Count
00	32 bits (DIV_NUMER)/32 bits(DIV_DENOM) Quotient 32 bits (DIV_RESULT), Remainder 32 bits (DIVREM_RESULT)	18 cycles
01	64 bits (DIV_NUMER)/32 bits (DIV_DENOM) Quotient 64 bits (DIV_RESULT), Remainder 32 bits (DIVREM_RESULT)	34 cycles
10	64 bits (DIV_NUMER)/64 bits (DIV_DENOM) Quotient 64 bits (DIV_RESULT), Remainder 64 bits (DIVREM_RESULT)	34 cycles

TWL-06-0017-001-D
Released: February 16, 2009

12.2 Square-Root Unit

The control registers that indicate the states of the square root calculator data register, calculation result register, calculation mode, and the square root calculator are shown below.

SQRT_PARAM: SQRT Data Register

Name: SQRT_PARAM Address: 0x040002B8 Attribute: R/W Initial value: 0x00000000_00000000

31	24	23	16	15	8	7	0
SQRT Data Lower Word							

31	24	23	16	15	8	7	0
SQRT Data Upper Word							

Unsigned integer (64-bit integer part)

SQRT_RESULT: SQRT Calculation Result Register

Name: SQRT_RESULT Address: 0x040002B4 Attribute: R/W Initial value: 0x00000000

31	24	23	16	15	8	7	0
SQRT Calculation Results Data							

SQRTCNT: SQRT Control Register

Name: SQRTCNT Address: 0x040002B0 Attribute: R/W Initial value: 0x0000

15							8	7							0
BUSY															MODE
Busy															Mode

- BUSY[d15]: Busy Flag

0	Square root calculator ready
1	Square root calculator busy

- MODE[d00]: SQRT Computation Mode

0	32-bit input
1	64-bit input

12.2.1 Number of Calculation Cycles

The SQRTCNT register's busy flag is set during the cycles shown in Table 12-2, according to the Computation Mode after writing to the Data Registers. When the busy flag has been cleared, you can find the calculation result by reading the register that stores the result.

Table 12-2 : Input Bit and Calculation Cycle Count by Computation Mode

SQRT Computation Mode	Input Bit Count	Calculation Cycle Count
0	32-bit input	13 cycles
1	64-bit input	13 cycles

13 Keys

TWL has A, B, L, R, +Control Pad, START, SELECT, X, and Y digital keys.

13.1 Input Keys

The status of the A, B, L, R, +Control Pad, START, and SELECT keys can be verified by reading the key input register (KEYINPUT) and checking the status of each bit. Because the X and Y keys are connected to the subprocessor, an API must be used to check the input status of these keys. The application can read all key data without regard for subprocessor operations when this API is used.

KEYINPUT: Key Input Register

Name: KEYINPUT Address: 0x04000130 Attribute: R Default Value: 0x0000

15						9	8	7							0
						L	R	DOWN	UP	LEFT	RIGHT	START	SEL	B	A
						Key Input									

- [d09–d00]: Key Input

0	Key is being pressed.
1	Key is not being pressed.

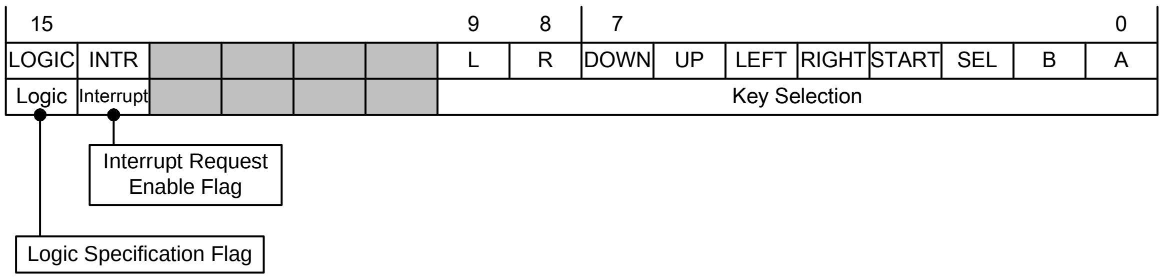
Note: ON-OFF may be repeated multiple times in a short time even if the user presses a key only once. To prevent a button from being pressed twice (chattering/switch bounce), allow an interval between readings of the key data (around 1 frame each). The input status of the X and Y keys cannot be read directly from the register.

13.2 Interrupt Handling for Key Input

Key input from the A, B, L, R, +Control Pad, START, and SELECT keys can generate interrupts. The key control register (KEYCNT) can be used to specify the key combinations and conditions for which interrupts can be generated.

KEYCNT: Key Control Register

Name: KEYCNT Address: 0x04000132 Attribute: R/W Default Value: 0x0000



- LOGIC[d15] : Logic Specification Flag

0	Detects if any of the specified keys was pressed
1	Detects if all of the specified keys were pressed

- INTR[d14] : Interrupt Request Enable Flag

0	Disable
1	Enable

- [d09–d00] : Key Selection

0	Key is not specified.
1	Key is specified.

Note: Interrupt handling cannot be specified for X or Y key input.

14 Sound

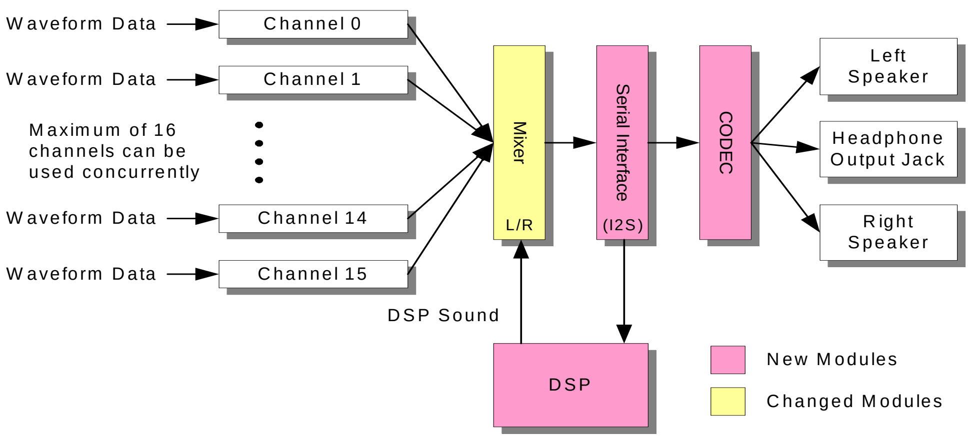
Like NITRO, TWL contains a sound circuit that enables the processing of 16 channels of simultaneous sound generation, two sound capture devices that can write the output from a specific channel or a mixer to memory, left and right speakers that can output sound, and a headphone output jack. The TWL also has a feature to mix DSP sounds generated in the DSP and a register volume feature for a mandatory shutter sound to be played when a photo is taken with the camera.

The specifications for the mixer have changed with TWL; whereas NITRO used PWM-format output, TWL uses I2S-format output. In addition, whereas output to the speakers and headphones were done with PMIC on NITRO, this has been changed to a module called “CODEC.”

While there have been changes to several of the modules, the TWL’s subprocessor (like NITRO’s subprocessor) carries out the sequence processing and the sound generation processing, so there will be no heavy burden on the main processor even when sound processing is performed.

The sound circuit is illustrated in Figure 14-1.

Figure 14-1 : Sound Circuit Outline Diagram



14.1 Hardware Specifications

The specifications for the included sound circuit hardware are as follows.

14.1.1 Data Format

8bitPCM, 16bitPCM, IMA-ADPCM, PSG rectangular waveforms, and noise can be used as formats for waveform data.

The 8bitPCM, 16bitPCM, and IMA-ADPCM data formats, as well as descriptions of PSG rectangular waveforms and noise are shown below.

14.1.1.1 8bitPCM Data Format

The 8bitPCM data format is shown below.

8bitPCM Data Format

31 24	23 16	15 8	7 0
Data 3	Data 2	Data 1	Data 0

14.1.1.2 16bitPCM Data Format

The 16bitPCM data format is shown below.

16bitPCM Data Format

31 24	23 16	15 8	7 0
Data 1		Data 0	

14.1.1.3 IMA-ADPCM Data Format

The header and data parts of the IMA-ADPCM data format are shown below.

IMA-ADPCM Header Format (First 32 bits)

31–24	23	22–16	15–8	7–0
		Table Index Initial Value	Initial Value	

IMA-ADPCM Data Format (From the 33rd bit)

31–28	27–24	23–20	19–16	15–12	11–8	7–4	3–0
Data 7	Data 6	Data 5	Data 4	Data 3	Data 2	Data 1	Data 0

Note: When repeatedly playing the ADPCM with the repeat feature, set the repeat pointer to the address of the data section, rather than that of the header section.

Also, if the repeat pointer is altered after starting playback on ADPCM, normal repeat play is not possible.

Be sure to stop playback before making changes to the repeat pointer.

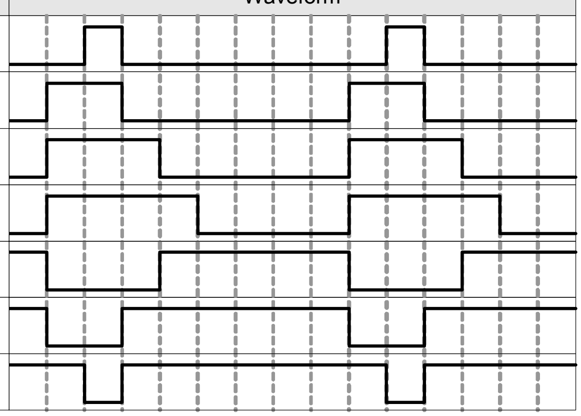
14.1.1.4 PSG Rectangular Waveforms

The *Programmable Sound Generator* (PSG) creates tones by altering the frequency of the output rectangular waveform (square waves) and the duty ratio.

The duty ratios of the PSG rectangular waveforms used on TWL can be altered as shown in Table 14-1.

Table 14-1 : Duty Ratio and PSG Rectangular Wave Waveforms

Duty Ratio	Waveform
12.5%	
25.0%	
37.5%	
50.0%	
62.5%	
75.0%	
87.5%	



14.1.1.5 Noise

Noise has no configurable items.

Noise can be used to generate white noise on a channel designated for noise.

14.1.2 Channels

Waveform data can be played simultaneously from 16 channels. However, PSG rectangular waveforms can be played on only 6 specific channels, and noise can be played on only 2 specific channels of the 16 channels.

Volume, pitch, and pan (orientation) can be configured for each channel.

The playable channels for each format are shown in Table 14-2.

Table 14-2 : Overview of Data Formats and Playable Channels

Data Format	Playable Channels
8bitPCM	Can be played on all channels from 0 to 15.
16bitPCM	
ADPCM	
PSG Rectangular Waveform	Can be played on Channels 8 to 13. 8bitPCM, 16bitPCM, and ADPCM cannot be played simultaneously on a channel that is playing a PSG rectangular waveform.
Noise	Can be played on Channels 14 to 15. 8bitPCM, 16bitPCM, and ADPCM cannot be played simultaneously on a channel that is playing noise.

14.1.3 Mixer

With NITRO, the mixer output used *pulse width modulation* (PWM), but this has been changed to I2S on TWL. I2S is a standard interface that is used to send and receive digital audio data between devices. On the TWL, the CPU and CODEC module are connected by an I2S bus, which is used to send and receive stereo audio data and microphone input audio data. On the TWL, the data length is fixed at 16 bits, and either 32.73 kHz or 47.61 kHz can be selected as the sampling rate.

14.1.3.1 Mixing with DSP Sounds

The TWL is capable of mixing DSP sounds that were generated in the DSP with NITRO sounds that are output from the sound circuit in the nine proportions shown in Table 14-3.

Table 14-3: Mixing Ratios for DSP Sounds

Configuration Pattern	NITRO Sound Ratio	DSP Sound Ratio	Comment
0	8 / 8	0 / 8	NITRO sound only
1	7 / 8	1 / 8	
2	6 / 8	2 / 8	
3	5 / 8	3 / 8	
4	4 / 8	4 / 8	
5	3 / 8	5 / 8	
6	2 / 8	6 / 8	
7	1 / 8	7 / 8	
8	0 / 8	8 / 8	DSP sound only

14.1.3.2 Register Volume

In addition to the standard analog sound volume whose audio volume can be changed, the TWL also has a “register volume” feature that ignores the analog sound volume and plays sound from the program at any volume. This is used, for example, to play a shutter sound when a photo is being taken with the camera.

14.1.3.3 CODEC

The function of the “CODEC” module connected to the TWL’s CPU is to perform conversion between analog and digital signals. In addition to converting I2S-format sound (a digital signal) output from the mixer into analog format and outputting it to the speakers or headphones, it also converts analog input from the microphone and Touch Panel into digital format.

Two operational modes have been prepared for CODEC. One is CODEC-DS, which was prepared for the sake of NITRO compatibility. The other is an improved CODEC-TWL mode. Applications must choose one of these operational modes.

The operational mode for CODEC is set by the IPL based on the information stored within the ROM. After the application is started, it is not possible to change the operational mode.

The “register volume” feature described above is only supported in CODEC-TWL mode. If it is necessary for your application to force a shutter sound to be played when a photo is taken, be sure to choose CODEC-TWL mode.

The differences between CODEC-DS mode and CODEC-TWL mode are summarized in Table 14-4.

Table 14-4: Differences Between CODEC-DS Mode and CODEC-TWL Mode

Category	Item	CODEC-DS Mode	CODEC-TWL Mode
Microphone Sampling-Related	Date Length (Amplitude Resolution)	8 bits or 12 bits	16 bits
	Sampling Rate	Variable	47.61 kHz / 32.73 kHz 23.81 kHz / 16.36 kHz 15.87 kHz / 10.91 kHz 11.90 kHz / 8.18 kHz
	Processing Load on Sub-Processor	Comparatively heavy (proportional to the sampling rate)	Comparatively light
	Gain Settings	26, 32, 38, 44 dB	0~59.5 dB (0.5 dB units)
Other	Register Volume Feature	Cannot be used	Can be used

14.1.4 Master Volume

The speaker output can be adjusted in 128 steps (from 0 to 127) via the master volume.

14.1.5 Sound Capture

There are two built-in sound capture devices on TWL that allow output waveform data to be written to memory.

Output from the left mixer or output from channel 0 can be written to memory with sound capture 0.

Output from the right mixer or output from channel 2 can be written to memory with sound capture 1.

The sampling frequency can be set up to 1.04876 MHz. The amplitude resolution can also be set from 8 bits to 16 bits.

14.1.6 Cautions

Playing waveform data with a high sampling rate, or playing sound whose pitch is higher than the original data because of fast-forwarding, leads to more frequent DMA transfers.

If DMA transfers occur frequently, they affect the main processor processes as well as subprocessor processes, such as wireless communications and the microphone.

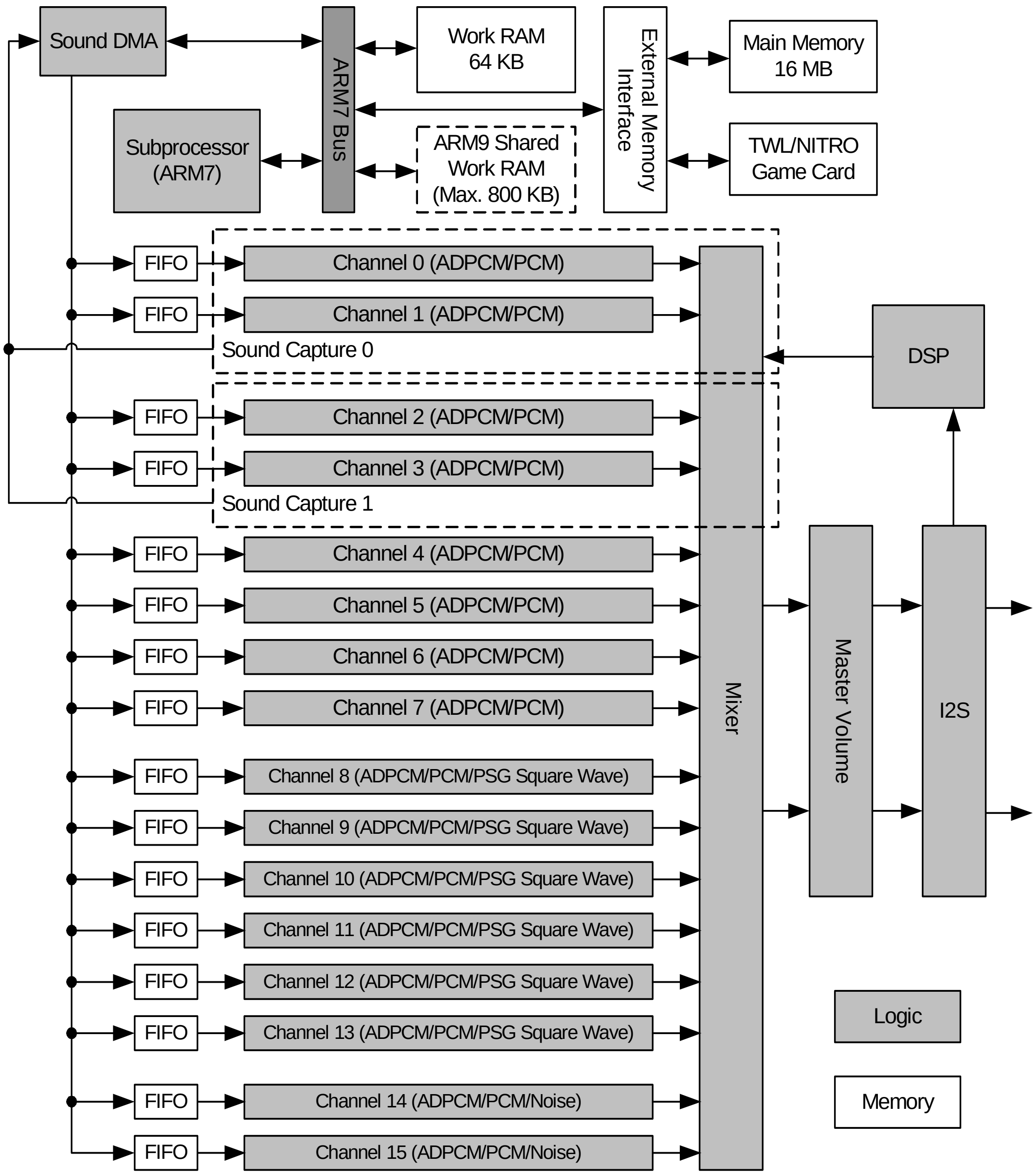
14.2 Sound Block Diagrams

This section shows the block diagrams for the TWL sound circuits.

14.2.1 Overall Sound

The block diagram for the overall sound circuit is shown in Figure 14-2.

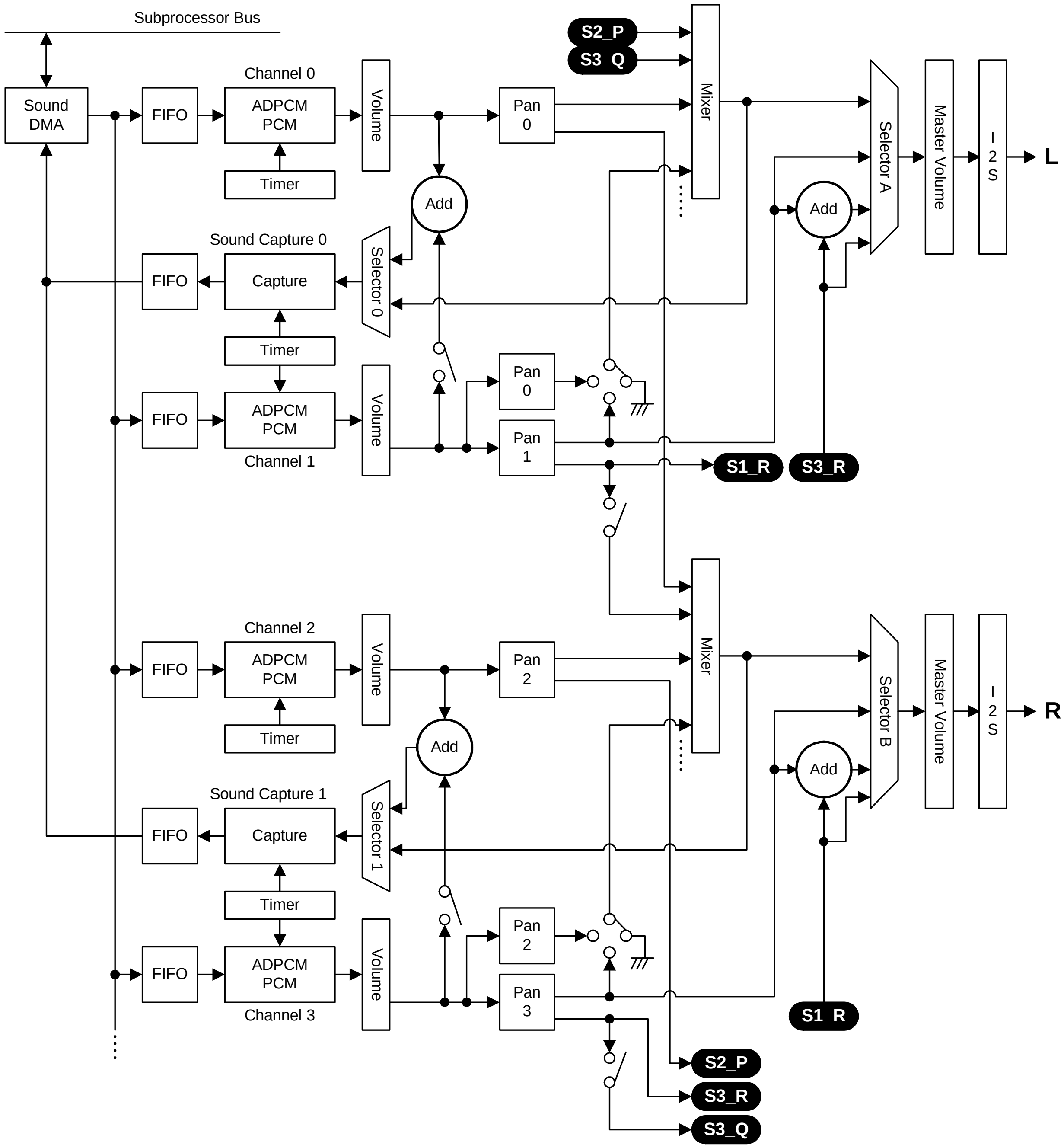
Figure 14-2 : Overall Sound Block Diagram



14.2.2 Channels 0 – 3 and Sound Capture 0 - 1

The block diagram for channels 0 to 3 and sound capture 0 and 1 is shown in Figure 14-3.

Figure 14-3 : Channel 0-3 and Sound Capture 0-1 Block Diagram



Caution 1: The switches before and after the pan blocks of channels 1 and 3 are always circuits connected to selectors A and B of the final-step output selection. Therefore, when channels 1 or 3 are selected as final-step outputs, sound will be output. Be aware of this when you do not want to output any sound.

Caution 2: The input signal to the mixer from the pan blocks of channels 1 and 3 is determined by the priority shown on Table 14-5. When channels 1 and 3 are set to bypass to the final-step output, the added channels are input to the mixer. Even if the output of that mixer is configured to be captured, channels 1 and 3 will not be input to the mixer (the switch immediately after the pan block will connect to GND, and the mixer input will always be 0 in this case only). As a result, there will be no reverb. Be aware that reverb using both the mixer and adder is not possible when channels 1 and 3 are set to bypass to the final-step output. If you are considering Caution 3 below, it is recommended that you use only the mixer.

Table 14-5 : Switch Input Priority from Channels 1 and 3 to the Mixer

Priority	Mixer Input	Switch State
High	0 Input (Connected to GND)	Sound Final-Step Output Bypass Configuration
	Pan 0 along with Pan 2	Channel Addition and Capture Configuration
Low	Pan 1 along with Pan 3	Normal Configuration

Caution 3: The TWL has two modes, TWL mode and NITRO-compatible mode. The following bug will occur in NITRO-compatible mode because fixes for the issue will not apply in NITRO-compatible mode.

There is a fault in the logic (preliminary to selector 0 and 1) for capturing the channel adder output. The following situations can occur:

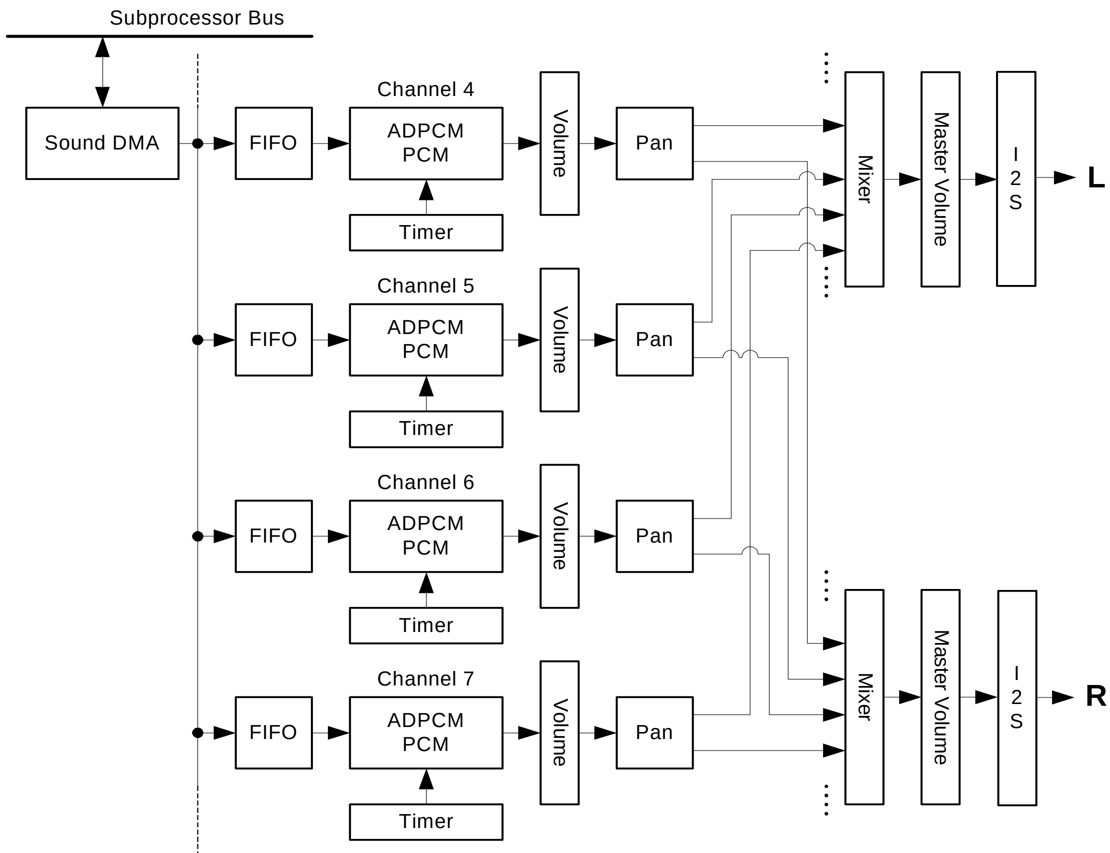
- When adding channels 0 and 1 along with channels 2 and 3, if an overflow occurs in either of the addition results, sign-inverted data will be output.
- When adding channels 0 and 1 along with channels 2 and 3, if the signs of channels 0 and 1 and channels 2 and 3 are each negative, the capture data will be forcibly converted to the maximum negative value.

Noise will be output in the sounds from these results. To deal with these faults, make sure that the addition data does not become saturated when using the adder, and that the values for channels 0-1 and channels 2-3 do not both become negative when not using the adder.

14.2.3 Channels 4 - 7

The block diagram for channels 4 to 7 is shown in Figure 14-4.

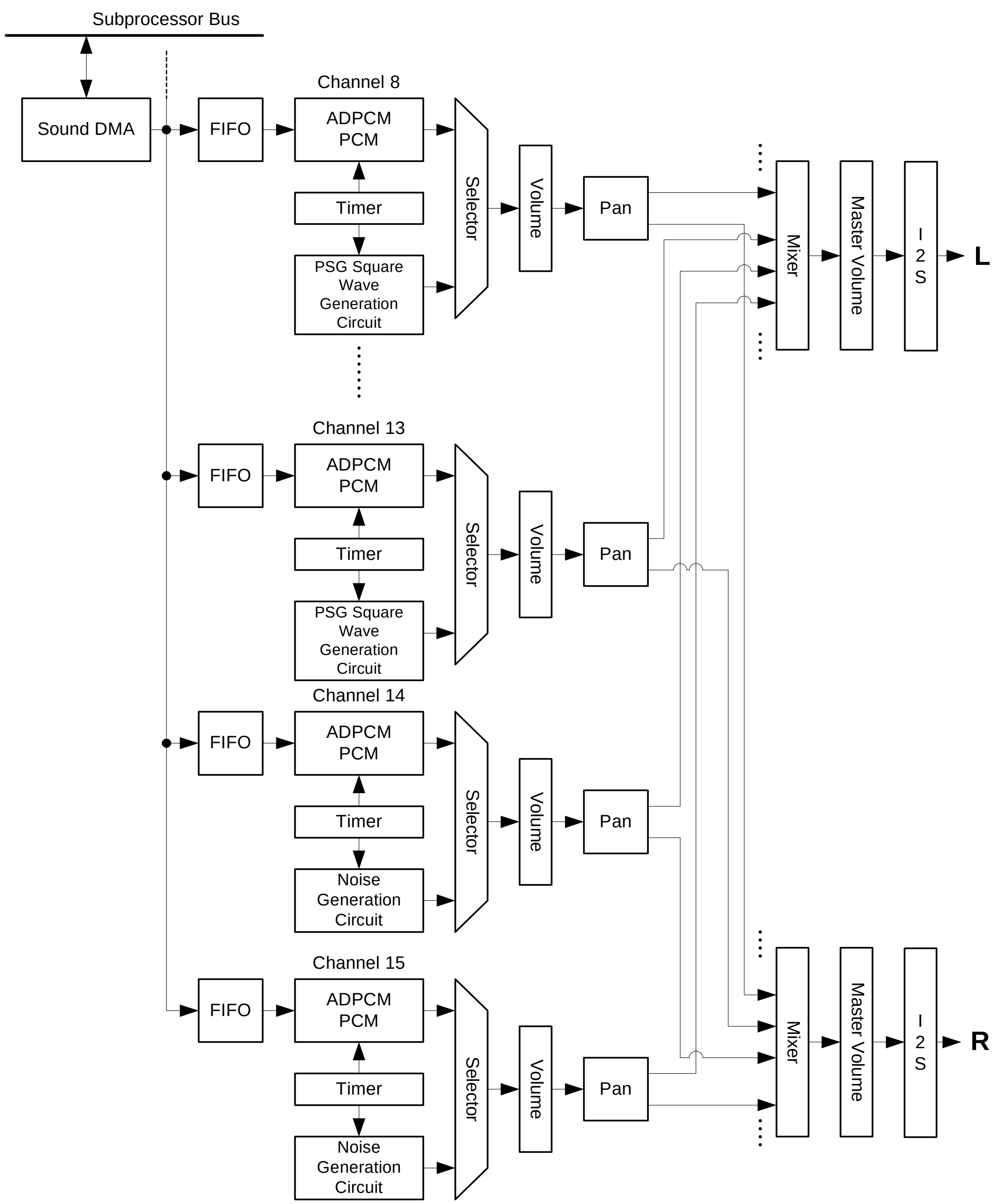
Figure 14-4 : Channel 4-7 Block Diagram



14.2.4 Channels 8 - 15

The block diagram for channels 8 to 15 is shown in Figure 14-5. PSG rectangular waveforms can be played on channels 8 to 13. Noise can be played on channels 14 and 15.

Figure 14-5 : Channel 8-15 Block Diagram



14.2.5 Examples of Using Sound

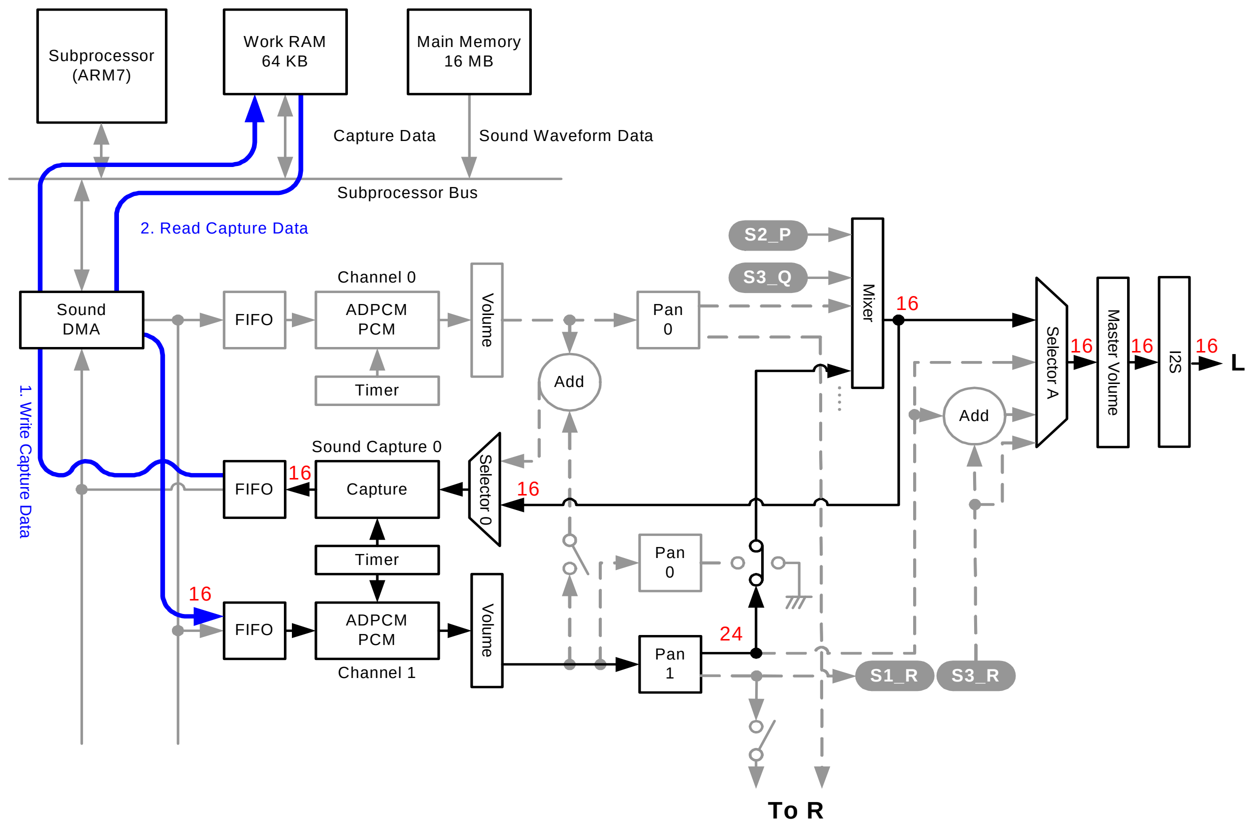
Examples of normal sound, a reverb effect using sound capture circuitry, and using the sound circuitry for sound effects are shown in the figures below.

14.2.5.1 Normal Use Example

Under normal use, the sound waveform data read from main memory is played on each channel, and then is output to the speaker via the mixer.

An example of a sound circuit under normal use is shown in Figure 14-6. Red numbers shown in Figure 14-6 indicate the bit count of data at the time of block input/output.

Figure 14-6 : Example of Sound Usage (Normal)



(Right Side Omitted)

14.2.5.2 Reverb Example

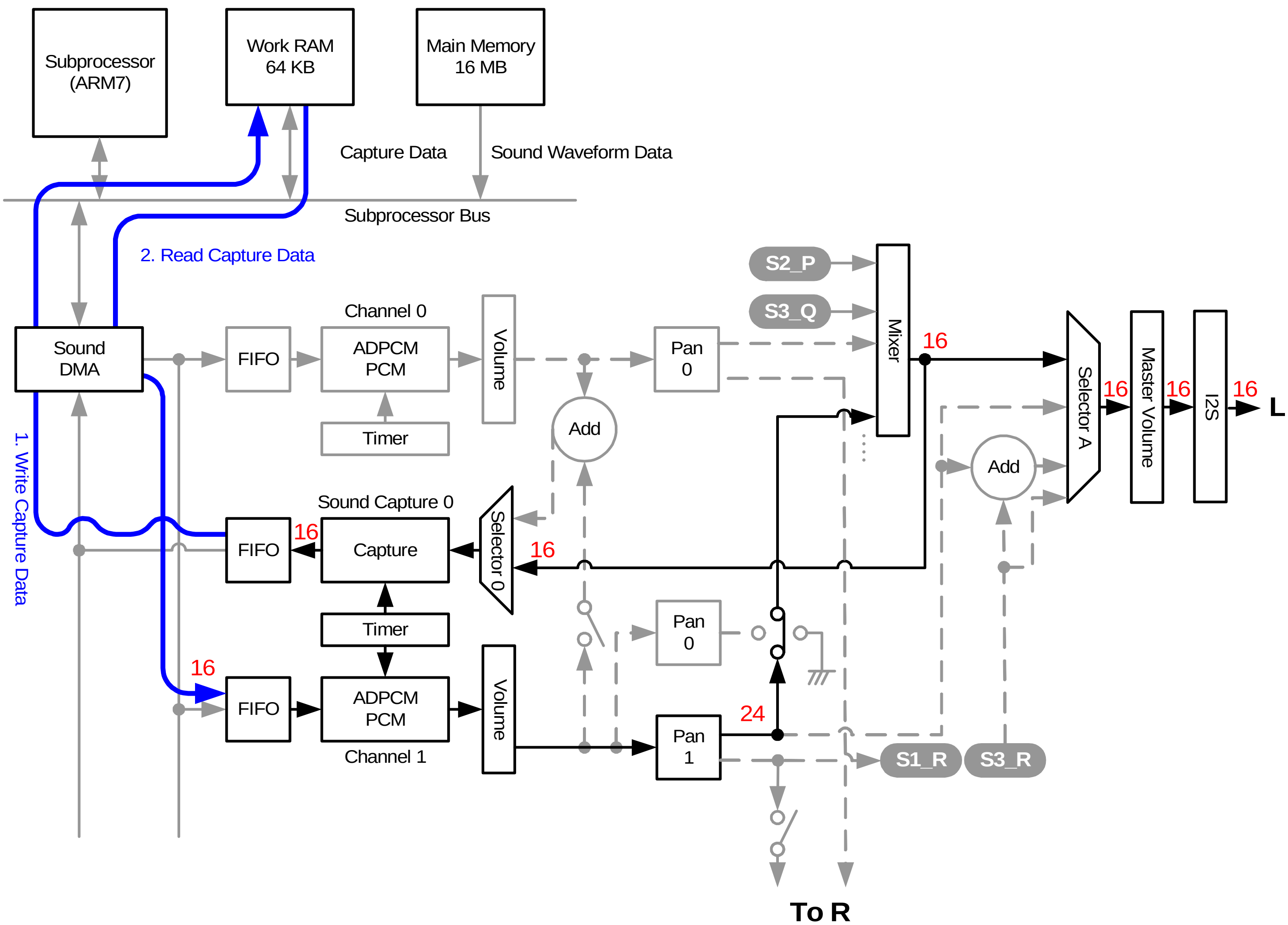
A reverb effect (echo) can be achieved using the sound capture feature.

Use sound capture to store the output from the mixer in Work RAM. A reverb effect results from playing the stored sound data on a channel and outputting it to the speakers via the mixer.

An example of sound circuit usage during reverb is shown in Figure 14-7. Red numbers shown in Figure 14-7 indicate the bit count of data at the time of block input/output.

In the example, Channel 1 is used for the reverb effect.

Figure 14-7 : Example of Sound Usage (Reverb)



(Right Side Omitted)

14.2.5.3 Effect Example

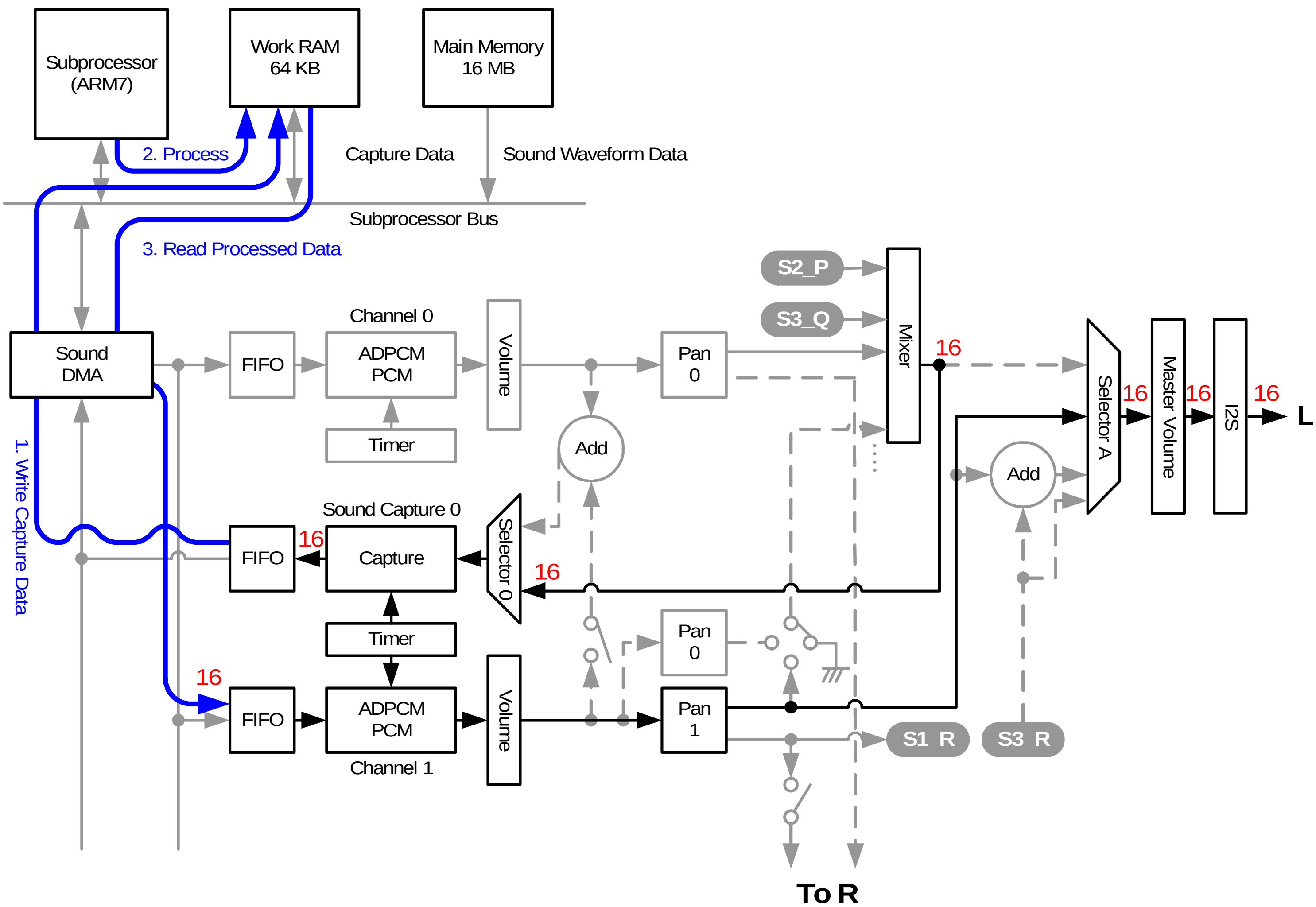
Data can be modified before outputting the sound by using the sound capture feature.

Store the capture data in Work RAM; after modifying the data using the subprocessor, output the data from a channel.

A sound effect example is shown in Figure 14-8. Red numbers shown in Figure 14-8 indicate the bit count of data at the time of block input/output.

In the example, Channel 1 is used for the effect.

Figure 14-8 : Example of Sound Usage (Effect)



(Right Side Omitted)

14.3 NITRO-Composer

There is no need to be concerned with the subprocessor operations when using the NITRO-Composer.

The NITRO-Composer allows easy playback of even complicated sounds, such as BGM.

14.3.1 NITRO-Composer Playback Method

There are three playback methods: sequence playback, waveform playback, and stream playback.

14.3.1.1 Sequence Playback

Sequence playback plays a variety of combined sounds, such as BGM and sound effects.

A maximum of sixteen sequences can be played simultaneously.

For example, during playback of a BGM that has one sequence assigned to it, up to fifteen sound effects can be played simultaneously.

A variety of parameters for each sequence, such as tempo and volume, can be individually controlled on the application side.

14.3.1.2 Waveform Playback

Waveform playback is a method for direct playback of waveform data.

It can play back waveform data, etc., that has been sampled with a microphone.

14.3.1.3 Stream Playback

Stream playback plays back long sequences, such as movie soundtracks.

Waveform data can be played back while the data stored on the TWL/NITRO card is sequentially loaded.

14.3.1.4 Latency Within the CODEC Module

During sound playback, there is a calculation latency of 21 / Fs (seconds) between when data enters the CODEC module and when the analog signal is output.

During microphone sampling, there is a calculation latency of 17/ Fs (seconds) between when data enters the CODEC module and when the result is output as data.

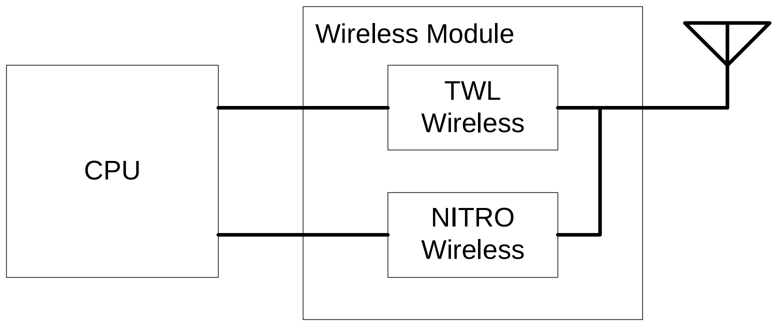
Note: “Fs” indicates the sampling rate (32.73 kHz / 47.61 kHz).

15 Wireless Communications

TWL contains two pieces of hardware for wireless communications using the 2.4GHz band. One is the same hardware built into the Nintendo DS and Nintendo DS Lite (NITRO wireless), and the other is new hardware that supports high-speed, high-security communication (TWL wireless).

It is only possible to use one of these wireless modules at a given time.

Table 15-1: Wireless Module



15.1 Hardware Specifications

The hardware specifications for wireless communications are shown in Table 15-2.

Table 15-2 : Wireless Communications Hardware Specifications

Item	Description	
	NITRO Wireless (NITRO-Compatible Wireless)	TWL Wireless
Band Used	2.4GHz band	Same as legacy
Communications Protocol	IEEE 802.11 (Internet Play) Nintendo's proprietary protocol (Multi-Card Play) Nintendo's proprietary protocol (Single-Card Play)	IEEE 802.11b/g (Internet Play)
Security	WEP 40 bit/128 bit compatible	WEP 40-bit/128-bit compatible WPA-PSK (TKIP/AES) WPA2-PSK (TKIP/AES)
Wireless Channels	13 channels	Depends on the country - USA/Taiwan: Channels 1-11 - Japan, Europe, Australia: Channels 1-13
Communications Speed	1 Mbps or 2 Mbps	1, 2, 5.5, or 11 Mbps 6, 9, 12, 18, 24, 26, 48, or 54 Mbps

Communications Range	10-30 meters This can change dramatically, depending on the environment and orientation of the unit.	Same as legacy
MAC Address	Unique for each TWL and thus can be used for identification.	Same as legacy
Interfering Devices	Devices that use the 2.4GHz band (cordless phones, microwave ovens, the wireless adapter for Game Boy Advance, WaveBird, other Wi-Fi devices, etc.)	Same as legacy

Note: When communications is used, power consumption also increases, so the battery will be consumed more quickly. Accordingly, when not using communication, put the unit in the STOP state using the wireless communications API.

The wireless adapter for Game Boy Advance cannot be used to communicate with NITRO.

Other devices may cause interference, making communications difficult. To avoid this, make the communications packet size as small as possible.

15.2 Wireless Manager

By calling the wireless manager API, you can control the wireless system without worrying about the operation of the subprocessor.

The unit can use Internet Play (it is also possible to select which wireless module to use), Multi-Card Play, and Single-Card Play.

15.2.1 Internet Play

In this mode, the unit can connect to the Internet using a wireless LAN (IEEE802.11b/g) access point.

15.2.2 Multi-Card Play

This mode allows wireless communication between a maximum of 16 systems.

Because it uses NITRO’s proprietary communications method, data units can be exchanged in less than one frame.

In Multi-Card Play, the maximum communications data size is 512 bytes.

15.2.3 Single-Card Play

This mode allows a child device without a TWL/NITRO card to download a game from a parent device with a TWL/NITRO card.

If the parent device sends data with a header that includes address information, the child device’s system ROM stores the data in the region specified by the header.

This mode uses Nintendo’s proprietary protocol.

Note: Games that are downloaded using DS Download Play (DS Single-Card Play) will run in NITRO mode.

16 Touch Panel

The lower screen includes a resistive-membrane touch panel that allows coordinates to be obtained in dot units. Although the touch panel can be operated with a finger, a touch pen with a 1.0 mm tip (see "Figure 16-1 : Comparison of LCD Dot Size and Touch Pen Size" on page 350) is included with the TWL as standard equipment.

The application can use the touch panel without regard for subprocessor operations when using the touch panel API. To overcome the differences in the coordinate position data on the API side, the calibration data stored in internal flash memory must be read with the IPL correction program and set with the API before using the touch panel.

The touch panel input data shown in Table 16-1 can be read with this API. The API reads in two ways: auto-sampling that reads four times per frame and request sampling that reads in real time in response to a request. When a request for a read is generated with request sampling, an interval of at least 4.17 msec must be maintained between requests to ensure a correct reading.

Table 16-1 : Touch Panel Input Data

Data Type	Data Description
Touch Panel Input Data	x coordinate (8 bit), y coordinate (8 bit) Touch Determination Flag (1 bit) Data Validity Flag (2 bit)

The details for input data for the touch panel are shown below.

- x-coordinate, y-coordinate

  x-coordinate : 0 - 255 (dots)
  y-coordinate : 0 - 191 (dots)

- Touch Determination Flag

  0 : The touch panel is not being touched
  1 : The touch panel is being touched

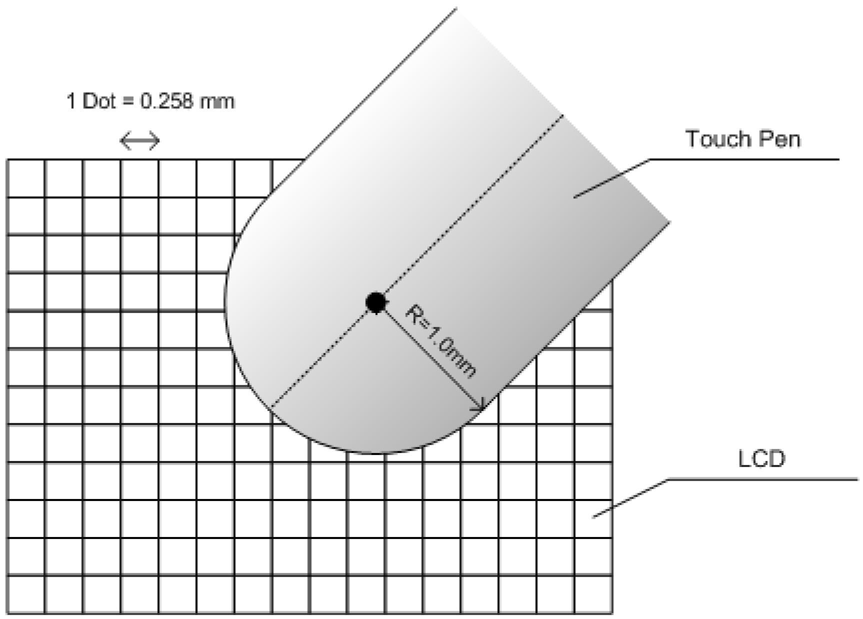
- Data Validity Flag

  00 : Both the x-coordinate and y-coordinate are valid
  01 : The x-coordinate is invalid
  10 : The y-coordinate is invalid
  11 : Both the x-coordinate and y-coordinate are invalid

Note: Structurally, the resistive-membrane touch panel can detect coordinates for only one location at a time. Therefore, if multiple locations are touched at the same time, the coordinates for each point cannot be detected. When the included stylus is used, the touch panel must be pressed down with a force of at least 80 g for the location to be detected. In some cases, the touch pen may not be able to depress areas within six dots of the screen edge as a result of built-in error between the touch panel and the TWL and limitations due to the shape of the touch pen tip.

On some occasions, irregular coordinate data are read immediately after the screen is touched or immediately after removing the touch pen from the screen. In these cases (in particular when operating a button displayed on screen), have the application use coordinates that are read in with the same value continually as valid data for processing. Note that the touch determination flag and the data validity flag are independent of each other, and that there may be a situation where, even though invalid data were stored, the touch panel is being touched. For example, cases may occur in which invalid data get stored while drawing a figure with a single stroke. In such cases, rather than determining that the user removed the touch pen from the touch panel, make the determination by reading the touch determination flag.

Figure 16-1 : Comparison of LCD Dot Size and Touch Pen Size

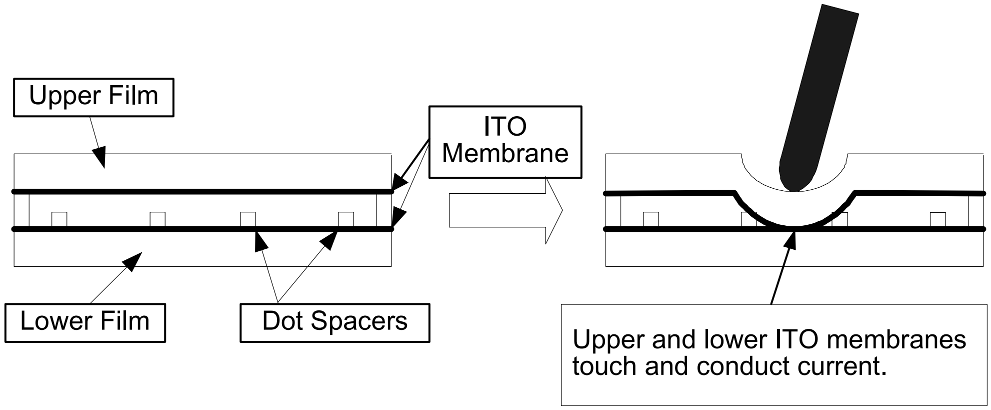


16.1 Touch Panel Structure

The construction of the resistive-membrane touch panel is illustrated in Figure 16-2.

Normally, the space formed between the upper and the lower films, both of which are coated with a transparent conducting membrane (ITO membrane: indium tin oxide), prevents current from being conducted. When a finger or stylus presses on the panel, the pressure causes the upper and lower films to touch and conduct current. The dot spacers prevent erroneous input and the NITRO from being continuously on.

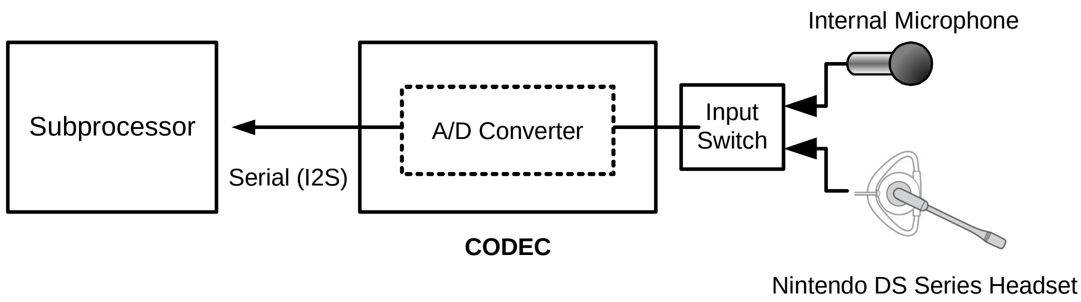
Figure 16-2 : Touch Panel Structure



17 Microphone

TWL is equipped with an omnidirectional condenser microphone that can be used for audio sampling. In addition, the user can use a Nintendo DS Series Headset. Microphone sensitivity can vary by up to a factor of two, depending on the console. Figure 17-1 shows a schematic diagram of the microphone.

Figure 17-1 : Microphone Schematic



17.1 Changes from NITRO

The analog-digital conversion that was done on the Touch Panel IC on NITRO, as well as the gain adjustment for the microphone input that was done on the PMIC, will be performed on the “CODEC” module due to a specification change for TWL.

Microphone input data converted with CODEC (in I2S format) is different from NITRO[1] in that the data length is fixed at 16 bits and the sampling rate can be changed to one of eight levels with 32.73 kHz / 47.61 kHz as the standard. Moreover, the gain can be adjusted within the range 10.5 - 70.0 dB in 0.5 dB units.

17.1.1 Operational Modes for CODEC

Two operational modes have been prepared for CODEC. One is CODEC-DS mode, which was prepared for the sake of NITRO compatibility. The other is the improved CODEC-TWL mode. Applications must select one of these operational modes.

The IPL sets CODEC’s operational mode based on information stored within the ROM. The mode cannot be changed after an application has started.

For more about the differences between CODEC-TWL mode and CODEC-DS mode, refer to "Table 14-4 : Differences Between CODEC-DS Mode and CODEC-TWL Mode" on page 336.

The microphone can be used without regard for subprocessor operations when the microphone API is used.

17.1.2 The Microphone in CODEC-TWL Mode

You can choose from one of four levels for the microphone sampling rate. The sampling rate will change depending on the combination with the sound sampling rate configured for the mixer.

1. On NITRO, the data length was either 8 bits or 12 bits, the sampling rate was variable, and the gain could be adjusted to one of four levels.

Table 17-1: Microphone Sampling Rates

Configuration Pattern	Sound Sampling Rate	
	47.61 kHz	32.73 kHz
0	47.61 kHz	32.73 kHz
1	23.81 kHz	16.36 kHz
2	15.87 kHz	10.91 kHz
3	11.90 kHz	8.18 kHz

Note: By using the microphone API, it is also possible to use sampling rates other than those shown in the table above.

The microphone gain can be specified in 0.5 dB increments in the range 10.5 - 70.0 dB.

17.1.3 The Microphone in CODEC-DS Mode

CODEC-DS mode follows the specifications used for the microphone on NITRO.

The sampling rates that can be specified range from several kHz to 32 kHz. However, the sampling rates for normal operation depend on the status of subprocessor use. The recording time depends on the memory size and sampling rate provided by the application.

The 60-Hz noise that is synched to the V-Blank is superimposed on the microphone input. However, this frequency is very low and the noise level is sufficiently low compared to audio input. Therefore, this will not cause a problem as audible sound.

Note: Wireless communication and sound processing also use the subprocessor. Therefore, if the microphone is used at the same time as these features, specify a sampling rate that takes into account the load on the subprocessor. In addition, the same serial bus is used to read touch panel data, access the internal flash memory, and control the PMIC. If the microphone is used at the same time as these features, ensure that conflicts do not occur. Do not use the microphone for 3 seconds after the power is turned on.

17.1.4 Microphone Input Values

The microphone input includes a noise component. As a result, Nintendo DS systems detect microphone input even when there is no sound. The microphone input values when there is no sound will vary from one system to another in the ranges indicated by Table 17-2 and Table 17-3.

To prevent a false determination that there is microphone input when there is none, avoid determining that there is microphone input within these ranges. Moreover, when the gain is high, the sound of the buttons being pressed and the sounds of the casing being rubbed will be more noticeably present in the microphone input. Keep this in mind when setting the threshold value.

Table 17-2: Microphone Input Values When There Is No Sound (For the Sound Circuitry of the Nintendo DS and the "CODEC-DS" Mode of the TWL System)

Amplitude Resolution		Noise Component (Range where it is prohibited to determine that there is microphone input)	Gain	
			Scale	dB
8 bit	Signed	±11	20	+26
		±13	40	+32
		±15	80	+38.1
		±20	160	+44.1
	Unsigned	117-139	20	+26
		115-141	40	+32
		113-143	80	+38.1
		108-148	160	+44.1
16 bit	Signed	±2368	20	+26
		±2880	40	+32
		±3392	80	+38.1
		±4672	160	+44.1
	Unsigned	30400-35136	20	+26
		29888-35648	40	+32
		29376-36160	80	+38.1
		28096-37440	160	+44.1

Table 17-3: Microphone Input Values When There Is No Sound (For the "CODEC-TWL" Mode of the TWL System)

Amplitude Resolution		Noise Component (Range where it is prohibited to determine that there is microphone input)	Gain	
			Scale	dB
8 bit	Signed	±3	3-20	10.5-26.0
		±5	21-40	26.5-32.0
		±9	42-80	32.5-38.0
		±16	84-160	38.5-44.0
		±32	168-320	44.5-50.5
		±51	355-640	51.0-56.0
		±110	669-1280	56.5-62.0
		±128	1334-2560	62.5-70.0
	Unsigned	125-131	3-20	10.5-26.0
		123-133	21-40	26.5-32.0
		119-137	42-80	32.5-38.0
		112-144	84-160	38.5-44.0
		96-160	168-320	44.5-50.5
		77-179	355-640	51.0-56.0
		18-238	669-1280	56.5-62.0
		0-256	1334-2560	62.5-70.0

Amplitude Resolution		Noise Component (Range where it is prohibited to determine that there is microphone input)	Gain	
			Scale	dB
16 bit	Signed	±768	3-20	10.5-26.0
		±1280	21-40	26.5-32.0
		±2304	42-80	32.5-38.0
		±4096	84-160	38.5-44.0
		±8192	168-320	44.5-50.5
		±13056	355-640	51.0-56.0
		±28160	669-1280	56.5-62.0
		±32768	1334-2560	62.5-70.0
	Unsigned	32000-33536	3-20	10.5-26.0
		31488-34048	21-40	26.5-32.0
		30464-35072	42-80	32.5-38.0
		28672-36864	84-160	38.5-44.0
		24576-40960	168-320	44.5-50.5
		19712-45824	355-640	51.0-56.0
		4608-60928	669-1280	56.5-62.0
		0-65535	1334-2560	62.5-70.0

Note: When using the TWL's microphone API to sample audio input with an effective amplitude resolution of 12 bits, the lower four bits are actually padded with zeros and 16 bits of data is obtained. Consequently, the 16-bit values in Table 17-2 and Table 17-3 are applicable when using 12-bit sampling.

Note: Do not use the amplitude level as the basis for determining whether microphone input is present when gain is set to +44.5 dB (168x) in CODEC-TWL mode, since the noise component will be quite noticeable in this case.

Note: TWL-compatible software that uses CODEC-TWL mode will run using the Nintendo DS sound circuitry when run on a DS or DS Lite system. As a result, be aware that the noise component will be different when run on a TWL than when run on a DS or DS Lite.

Not all systems are guaranteed to pick up the full range of microphone input values that can be expressed with the number of bits for a given amplitude resolution. The individual variation between systems is shown in Table 17-4 and Table 17-5. In determinining whether microphone input is present, avoid methods that rely on values outside the range of guaranteed input values.

Table 17-4: Guaranteed Microphone Input Ranges (For the Sound Circuitry of the Nintendo DS and the "CODEC-DS" Mode of the TWL System)

Amplitude Resolution		Outside of Guaranteed Input Range LOWER LIMIT	Outside of Guaranteed Input Range UPPER LIMIT	Guaranteed Microphone Input Value Ranges
8 bit	Signed	-128 to -101	+100 to +127	-100 to +99
	Unsigned	0 to 27	228 to 255	28 to 227
16 bit	Signed	-32768 to -25664	+25648 to +32752	-25663 to +25647
	Unsigned	0 to 7104	58416 to 65520	7105 to 58415

Table 17-5: Guaranteed Microphone Input Ranges (For the "CODEC-TWL" Mode of the TWL System)

Amplitude Resolution		Outside of Guaranteed Input Range LOWER LIMIT	Outside of Guaranteed Input Range UPPER LIMIT	Guaranteed Microphone Input Value Ranges
8 bit	Signed	-128 to -124	+123 to +127	-123 to +122
	Unsigned	0 to 4	251 to 255	5 to 250
16 bit	Signed	-32768 to -31744	+31743 to +32752	-31743 to +31742
	Unsigned	0 to 1024	64512 to 65520	1025 to 64511

Note: There may be feedback howl and incorrect playback, depending on the system, if the recording of the sound input from the microphone and playback of that recorded sound are performed at the same time.

Note: If using the TWL's microphone API to sample audio input with an effective amplitude resolution of 12 bits, the lower four bits are actually padded with zeros and 16 bits of data is obtained. Consequently, the 16-bit values in Table 17-4 and Table 17-5 are applicable when using 12-bit sampling.

18 Real-Time Clock (RTC)

The TWL has an internal Real-Time Clock (RTC). Time is kept by means of an auto-calendar feature that extends through 2099 and accounts for leap years. The maximum error for the clock is ±4 seconds/day.

The time is set on the following occasions:

1. When the power is turned on for the first time after the unit is purchased.
2. When the power is turned on after changing the battery.
3. When the unit is restarted after the batteries have been drained (unit has been sitting for several months with no charge in the batteries).
4. When the date and time are set from the boot menu.

When the RTC API is used, the RTC can be used without regard for subprocessor operations. The real-time data shown in Table 18-1 can be read with this API.

Table 18-1 : Real-Time Data

Data Type	Data Description
Real-Time Data	Year (00 - 99), month (01 - 12), date (01 - 31), day (00 - 06), hours (00 - 23), minutes (00 - 59), seconds (00 - 59) Each value is stored as a binary coded decimal (BCD) value.

In addition, two types of alarm functions are provided. By setting the alarm from the application and engaging sleep mode, the unit can be awakened from sleep mode at a specified time. For information on sleep mode, see "11 Power Management" on page 317. The API can be used to read and write the following settings for Alarm 1 and Alarm 2.

Table 18-2 : Settings for Alarm 1 and Alarm 2

Data Type	Data Description
Alarm 1 and Alarm 2	Year (00 - 99), month (01 - 12), date (01 - 31), day (00 - 06), hour (00 - 23), minutes (00 - 59) Separate settings for alarm 1 and 2 can be specified. Each value is stored as a binary coded decimal (BCD) value. Day, hour, and time can be enabled or disabled. Example: The alarm can be set to activate at the same time every day by disabling the day setting.

Note: Real-time data cannot be written from the application to the RTC.

19 Internal Flash Memory

The TWL has internal flash memory to store touch panel calibration data, profile, the TWL initial settings data, and RTC operation information data. The internal flash memory is dedicated memory for the storage of TWL settings data; the application cannot write to the internal flash memory.

19.1 Touch Panel Calibration Data

This calibration data is used to compensate for the variation in coordinate positioning data between individual touch panels. If an application uses the touch panel, the touch panel must be set by reading the calibration data with the API.

19.2 Profile Data

The profile data stores information about the owner of the TWL. The profile data in Table 19-1 can be read using the API.

Table 19-1 : Profile Data

Data Type	Data Contents
Owner Information Data	User ID (22 bytes), User color (1 byte), Birthday (2 bytes), Comment (46 bytes)

The details of the profile data are shown below.

- User ID (Nickname)

  Nickname string : Maximum of 10 Unicode (UTF-16) characters (20 bytes). No termination code.

  String length : Nickname string length (2 bytes).

- User Color (Favorite color)

  0-15 : Selected from a set of 16 colors determined by IPL. (RGB values are enclosed by parentheses.)

0: GRAY	(12,16,19)	1: BROWN	(23, 9, 0)
2: RED	(31, 0, 3)	3: PINK	(31,17,31)
4: ORANGE	(31,18, 0)	5: YELLOW	(30,28, 0)
6: LIME GREEN	(21,31, 0)	7: GREEN	(0,31, 0)
8: DARK GREEN	(0,20, 7)	9: SEA GREEN	(9,27,17)
10: TURQUOISE	(6,23,30)	11: BLUE	(0,11,30)
12: DARK BLUE	(0, 0,18)	13: PURPLE	(17, 0,26)
14: VIOLET	(26, 0,29)	15: MAGENTA	(31, 0,18)

- Birthday (month and day) (Each is stored as a binary-coded decimal number.)

  Month (1 byte) : Month of birth (01-12)

  Day (1 byte) : Day of birth (01-31)

- Comment

  A comment of two lines of a maximum of 23 characters each (23 bytes x 2 = 46 bytes) using Unicode (UTF-16).

19.3 TWL Initial Settings Data

The TWL initial settings data stores the language setting.

Using the API, the TWL initial settings data in Table 19-2 can be read.

Table 19-2 : TWL Initial Settings Data

Data Type	Data Content
TWL Initial Settings Data	Language setting (4 bytes)

The details of the TWL initial settings data are shown below:

- Language setting

  0 : Japanese

  1 : English

  2 : French

  3 : German

  4 : Italian

  5 : Spanish

19.4 RTC Operation Information Data

When setting the real-time clock (RTC), the difference in seconds with the initial data is set. This information can be used to determine if the user has changed the RTC data.

Using the API, the RTC operation information data in Table 19-3 can be read.

Table 19-3 : RTC Operation Information Data

Data Type	Data Content
RTC Operation Information Data	RTC offset value. This value changes each time the RTC setting is changed.

20 AES Engine

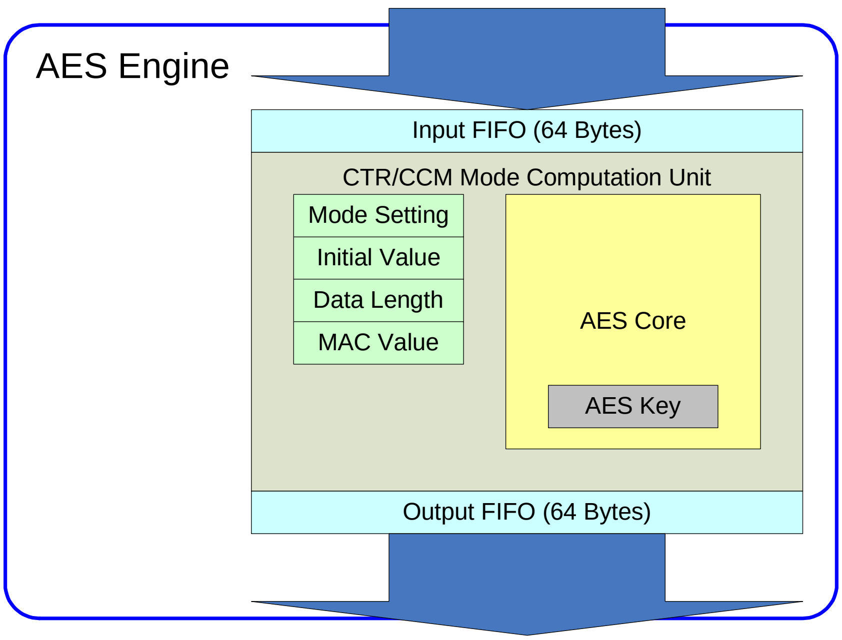
The TWL system includes an AES (Rijndael) encryption and decryption engine.

AES (Advanced Encryption Standard) is a symmetric-key (shared-key) type block cipher that has been adopted as an encryption standard in the United States. For details about AES, refer to *Federal Information Processing Standards Publication 197*.

Following is an overview of the built-in AES engine.

- Encryption Mode: CTR (counter) mode / CCM (counter with CBC-MAC) mode
  (Both modes support both encryption and decryption.)
- Key Length: 128 bits
- Block Size: 128 bits

Figure 20-1 : The AES Engine

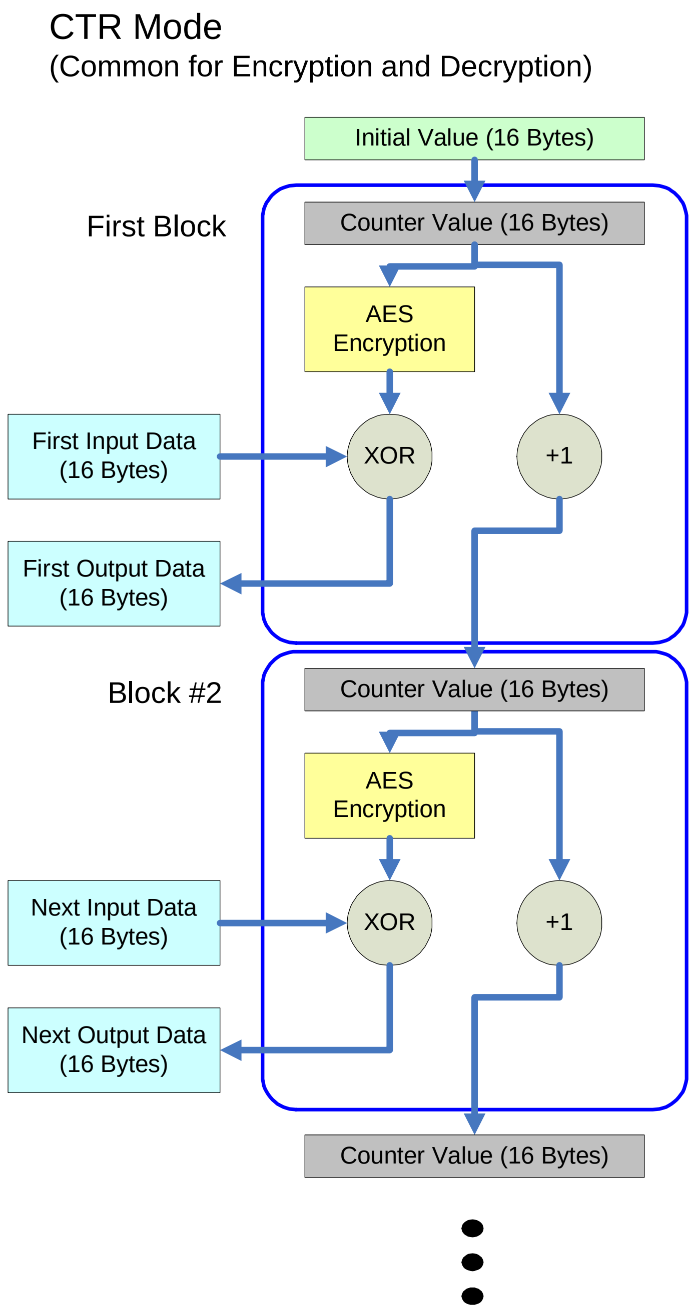


All AES operations should be done via the API.

20.1 CTR Mode

CTR mode is a type of block cipher algorithm (see Figure 20-2). For details, see section 6.5 of *NIST SP800-38A*. One of the limitations of this AES engine is that the input and output data must be in 16-byte units.

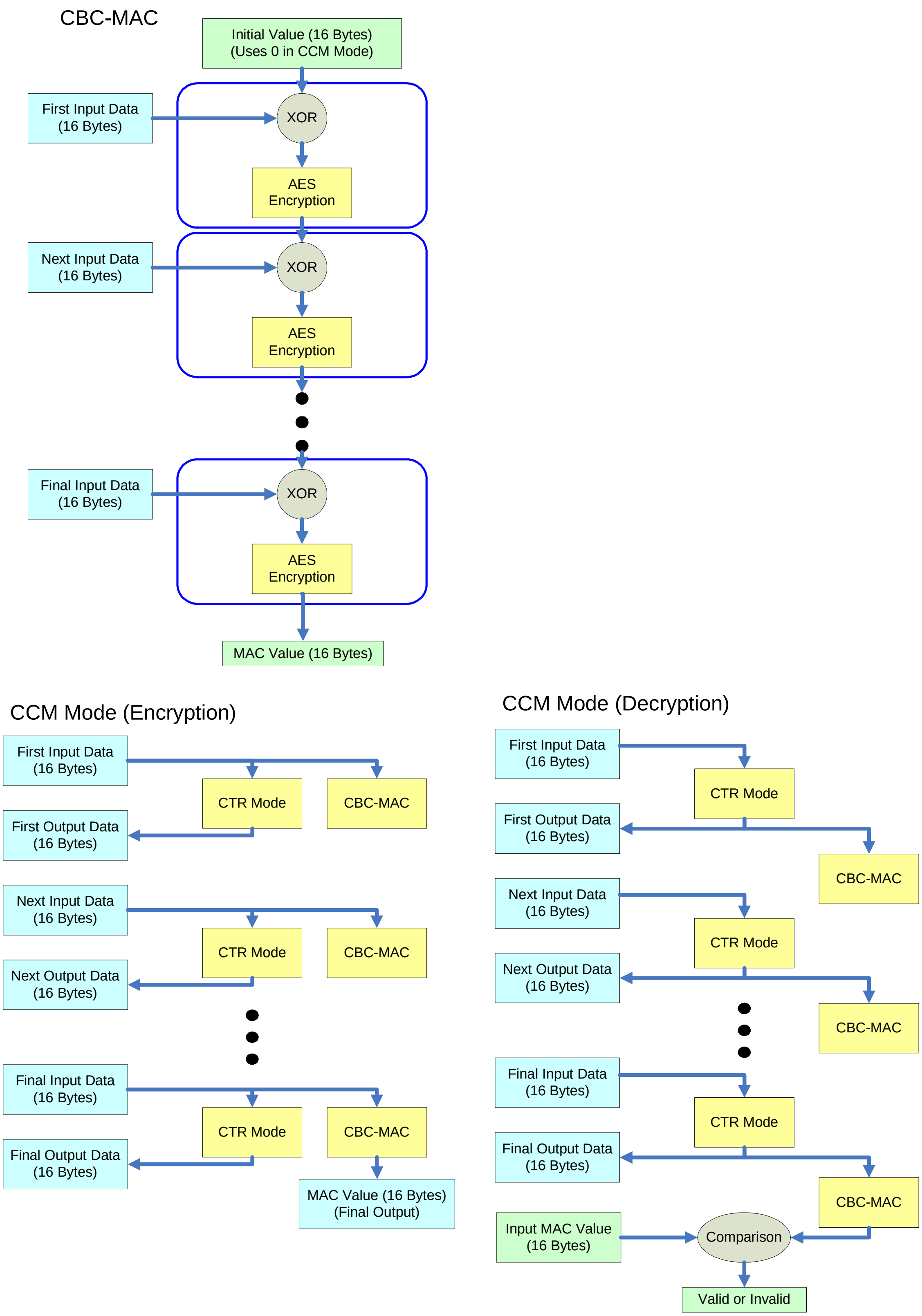
Figure 20-2 : CTR Mode



20.2 CCM Mode

CCM mode is a type of block cipher algorithm that combines CTR mode with CBC-MAC, which uses the final output data from CBM mode as the MAC value (see Figure 20-3). For details, see *NIST SP800-38C*.

Figure 20-3 : CCM Mode



It is also possible to enter data (“associated data”) that is not encrypted but is used for the CBC-MAC calculation separately from the data to be encrypted or decrypted (the “payload”). When doing so, the associated data is entered before the payload.

In CCM mode, the AES engine also generates the first block (B0) internally in advance of the associated data and payload. The method for generating the first block (B0) was implemented as the case where q=3 (fixed) described in Appendix A of *NIST SP800-38C*.

Due to these specifications, the size that can be specified as Nonce is 96 bits.

Also, the engine doesn’t perform any special processing such as adding the size information. If such information is required, enter the pre-formatted information as associated data.

A limitation of this AES engine is that the input and output data, associated data, and payload must be in 16-byte units.

One of the features of this AES engine is that it is possible to suppress output of associated data.

20.3 Precautions

In most AES libraries, the key data, MAC data, and input/output data are calculated as big-endian data. In contrast, this AES engine treats all data as 128-bit or 96-bit little-endian data, so be aware of this issue.

21 Camera

The TWL system includes two cameras. Although the cameras can be controlled independently, it is not possible to obtain images from both cameras simultaneously. Camera operations should all be done via the API.

The basic specs of the camera are shown below.

- Aperture: *f*/2.8 (Fixed)
- Angle of view (when photographing at maximum resolution)
  Diagonal: 66 degrees
  Horizontal: Approximately 54 degrees
  Vertical: Approximately 42 degrees
- Depth of field: 20cm - Infinity (deep focus; camera does not contain a macro switch)
- Maximum resolution: VGA
- Maximum frame rate: 30 fps
- Output format: YC_rC_b (a YUV>RGB conversion circuit also makes it possible to output in RGB555)
- Resolution settings (supports fast configuration changes)
- Frame rate settings
- Effect processing (supports fast configuration changes)
- Flipping (supports fast configuration changes)
- Photo mode settings
- White balance settings
- Exposure settings
- Sharpness settings

Of these specifications, two settings can be maintained for each camera for the resolution, effect processing, and flipping operations. Switching between the two settings can update the settings faster than setting them individually.

21.1 Camera Settings

The various camera settings that can be configured, such as the resolution and framerate, are shown below.

- Resolution settings
  - VGA (640 pixels wide, 480 pixels high)
  - QVGA (320 pixels wide, 240 pixels high)
  - QQVGA (160 pixels wide, 120 pixels high)
  - CIF (352 pixels wide, 288 pixels high)
  - QCIF (176 pixels wide, 144 pixels high)
  - DS screen zize (256 pixels wide, 192 pixels high)
  - Doubled height and width of DS screen size (512 pixels wide, 384 pixels high)
- Frame rate settings
  - 15 fps, fixed
  - 15-5 fps, variable
  - 15-2 fps, variable
  - 8.5 fps, fixed
  - 5 fps, fixed
  - 20 fps, fixed
  - 20-5 fps, variable
  - 30-5 fps, variable
  - 30 fps, fixed
- Effect processing
  - Monochrome - Converts the image to black and white.
  - Sepia tone (yellow) - Gives the image a sepia tone. Applies an overall yellow ochre color and gives images a nostalgic feel.
  - Sepia tone (red) - Gives the image a sepia tone. Applies an overall auburn color and gives images a nostalgic feel.
  - Inversion - Inverts tone and color, and applies a bluish-violet tone.
  - Inversion (film tone) - Inverts tone and color, and applies a magenta tone. The U and V components in the inverted image's YUV data are in reverse order.
- Flipping
  - Fip vertical
  - Flip horizontal
  - 180° rotation
- Photo mode settings
  - Text capture (QR codes, letters)

    Increases the sharpness and contrast in order to emphasize text outlines. This can capture relatively detailed text such as business cards, newspapers, and examination sheets in a way that makes recognition easier than normal. QR codes can also be used to some degree.
  - Portrait

    Reduces the contrast and lowers the emphasis on outlines relative to the standard setting.
  - Landscape

    Reduces the exposure (AE) levels below the standard setting and keeps the white balance fixed to the Daylight setting, under the assumption that photos will be taken outdoors in this mode.

  - Night View

    Increases the gain so that images can be captured at a high frame rate, even in low light conditions. The noise component of the images will be increased proportionately. Slightly reduces the sharpness relative to the standard setting. Uses the standard automatic mode for white balance. Taking pictures in this mode under well lit conditions will cause the amount of image noise to increase.

  - Night Snap

    This mode has not been prepared as a unique mode, but by combining Landscape mode with the variable 15-2 fps frame rate setting, the camera can be made to take pictures in a “night scenery” mode. The camera will not operate correctly if the white balance is set to “auto” under low light conditions, so this mode keeps the white balance fixed to the Daylight setting. Also, because lower frame rates will have less image noise, this mode uses the variable 15-2 fps frame rate setting in order to keep the frame rate low under low-light conditions. The sharpness in this mode is slightly higher than the standard setting.

- White balance settings
  - Standard (Automatically adjusted)
  - Tungsten (3200K)
  - White Fluorescent Light (4150K)
  - Daylight (5200K)
  - Cloudy/Horizon (6000K)
  - Shade (7000K)

  Note: The automatic white balance (AWB) can be temporarily disabled as well.

- Exposure settings

  Can be configured from -5 to +5.

  Note: The automatic exposure (AE) can be temporarily disabled as well.

- Sharpness settings

  Can be configured from -3 to +5.

- RGB conversion

  Image data from the camera can be obtained in either YC_rC_b (YUV422) format or RGB (RGB555) format.

  The formulas for converting YUV422 to RGB888 are shown below:

$$R = 1.000 \times Y + 1.402 \times (V - 128)$$
$$G = 1.000 \times Y - 0.344 \times (U - 128) - 0.714 \times (V - 128)$$
$$B = 1.000 \times Y + 1.772 \times (U - 128)$$

  The results of the calculation will be rounded off if they exceed 255. Then they will be scaled to the range [0-31], after which they will be in RGB555 format.

  The formulas for the reverse conversion (i.e., converting RGB555 to YUV422) are shown below:

$$Y = 0.299 \times R + 0.587 \times G + 0.114 \times B$$
$$U = -0.169 \times R - 0.331 \times G + 0.500 \times B + 128$$
$$V = 0.500 \times R - 0.419 \times G - 0.081 \times B + 128$$

21.2 Trimming

It is possible to trim (crop) a portion of a captured image.

21.3 DMA Transfer

The new DMA that was added with the TWL is always used in order to transfer images captured by the camera to main memory. During DMA transfers, the transfer size for a single transfer must be set to 2 KB or less (1024 pixels, since 1 pixel = 2 bytes).

21.4 Recovery from Sleep Mode

When transitioning to Sleep mode, the cameras will automatically go into standby. If you want to enable the cameras after recovering from Sleep mode, you must handle this on the application side.

22 DSP

TWL includes an XpertTeak™ core made by CEVA, Inc. (the ™ notation will be omitted throughout the remainder of this document) as well as a DSP (Digital Signal Processor) block with added local memory, which can be shared by the main processors and subprocessors as work RAM.

22.1 CEVA-Teak Core

The CEVA-Teak core has dual MAC (Multiply and Accumulation), a word size of 16 bits, and a DSP core with 16-bit instructions. It has the following characteristics:

- Four 40-bit accumulators
- Two 16x16-bit -> 32-bit multipliers
- Multiplication and addition can be run in parallel in a single cycle
- Zero-overhead loops

It is also equipped with peripheral circuitry such as an 8-channel DMA and an interrupt controller. Several communication interfaces have been prepared between the core and the main processor.

22.2 DSP Memory

The 16 x 32 KB memory planes built into the DSP core can be used both as local memory for the DSP core and as work RAM for the main processor and subprocessor.

By configuring the registers, it is possible to specify which systems can access each memory plane (it is not possible to have multiple cores access the same memory plane simultaneously), and the contents of the memory will be retained even if the master (a system that can access the memory plane) is switched.

Memory mapping and switching the master are done using the System Control register. For details, see "Work RAM" on page 40.

22.3 Communication with the Main Processor

The DSP core can be controlled from the main processor using the interface register. This interface has the following features.

- Resetting the DSP
- Reading data from and writing data to the DSP's internal memory space (DSP memory transfer)
- Sending and receiving small amounts of data using the command / reply register
- Using the semaphore register to handle exclusion control between the ARM9 and DSP and generate interrupts

22.3.1 Reset

The DSP can be reset from the main processor by manipulating the DSP's configuration register (DSP_PCFG). This reset operation will initialize the DSP core and its peripheral circuitry, but the interface register itself will not be reset.

The configuration register also makes it possible to select the memory space to be accessed during a DSP memory transfer, provide interrupt notifications using the status of the read and write FIFOs, and configure the data length for read operations.

DSP Configuration Register

Name	Address	Attribute	Initial Value
DSP_PCFG	0x04004308	R/W	0x0000

15–12	11	10	9	8	7	6	5	4	3–2	1	0
MEMSEL	PRIE2	PRIE1	PRIE0	WFEIE	WFFIE	RFNEIE	RFFIE	RS	DRS	AIM	DSPR

- MEMSEL[d15-d12] : Memory Select

  Selects the memory to access during a read from or write to the DSP's internal memory space (DSP memory transfer).

	Effect of Setting
0000	Data memory
0001	Peripheral (MMIO) register
0101	Program memory (only write access)

- PRIE0-2[d09],[d10],[d11] : Reply Register Write Notification Flag

  When set to 1, the main processor will be notified of an interrupt when the DSP core writes data to the corresponding reply register (DSP_REPx).
- WFEIE[d08] : Write FIFO Empty Notification Flag

  When set to 1, the main processor will be notified of an interrupt when the write FIFO becomes empty.
- WFFIE[d07] : Write FIFO Full Notification Flag

  When set to 1, the main processor will be notified of an interrupt when the write FIFO becomes full.
- RFNEIE[d06] : Read FIFO Not Empty Notification Flag

  When set to 1, the main processor will be notified of an interrupt when the read FIFO is no longer empty.
- RFFIE[d05] : Read FIFO Full Notification Flag

  When set to 1, the main processor will be notified of an interrupt when the read FIFO becomes full.
- RS[d04] : DSP Memory Read Start Flag

  When a 1 is written, read access to the DSP memory space will begin (the read FIFO will be cleared at this time).

  When a 0 is written, read access that is under way is cut off.
- DRS[d03-d02]: DSP Memory Read Data Length

  Specifies the length of data to read during read access.

	Effect of Setting
00	Single access (1 word)
01	8-word burst read
10	16-word burst read
11	Free-running (read access will continue until it is cut off)

- AIM[d01] : Address Auto-Increment Mode

  When set to 1, the transfer address will automatically be incremented after each DSP memory transfer.

- DSPR[d00] : DSP Reset

  When set to 1, the DSP core and its peripheral circuitry will be reset.Once it has been set to 1, do not write a 0 for at least 8 cycles of the DSP clock.

22.3.2 Status Register

The status register (DSP_PSTS) lets you know various statuses during communication between the DSP and the main processor.

If you have configured the system so that the main processor will not be notified of interrupts from the DSP during DSP memory transfers, you can look the bits in this register (WFEI, WFFI, RFNEI, and RFFI) to find out the status of the read and write FIFOs.

DSP Status Register

Name	Address	Attribute	Initial Value
DSP_PSTS	0x0400430C	R	0x0000

15			12	11	10	9	8	7	6	5	4	3	2	1	0
RCMD IM2	RCMD IM1	RCMD IM0	PRI2	PRI1	PRI0	PSEMI	WFEI	WFFI	RFNEI	RFFI			PRST	WTIP	RTIP

- RCMDIM0-2[d13],[d14],[d15] : Command Register Read Flag

  Indicates whether the DSP has read a value written to the DSP command register (DSP_CMDx) by the main processor.

	Meaning
0	The DSP has read the command register
1	The DSP has not read the command register

- PRI0-2[d10],[d11],[d12] : Reply Register Update Flag

  Indicates whether the DSP has updated the DSP reply register (DSP_REPx).

	Meaning
0	The DSP has updated the reply register
1	The DSP has not updated the reply register

- PSEMI[d09] : Semaphore Register Interrupt Request Flag

  When 1, indicates that there is an interrupt request from the DSP through the semaphore register (DSP_SEM) from the DSP (a 1 was written to an unmasked bit in the DSP semaphore register DSP_PMASK).

- WFEI[d08] : Write FIFO Empty Flag

  When 1, indicates that the write FIFO is empty.

- WFFI[d07] : Write FIFO Full Flag

  When 1, indicates that the write FIFO is full.

- RFNEI[d06] : Read FIFO Not Empty Flag

  When 1, indicates that the read FIFO is not empty.At least one read can be performed from the DSP transfer data register (DSP_PDATA).

- RFFI[d05] : Read FIFO Full Flag

  When 1, indicates that the read FIFO is full.

- PRST[d02] : Peripheral Reset Flag

  When 1, indicates that the peripheral (circuitry) is in the process of being reset. After a reset is issued to the DSP, once you can verify that this bit has become 0, the peripheral circuitry can be accessed while the DSP core is still in the reset state.

- WTIP[d01] : Write Transfer Underway Flag

  When 1, indicates that a write transfer to the DSP memory is under way.

- RTIP[d08] : Read Transfer Underway Flag

  When 1, indicates that a read transfer from the DSP memory is under way.

22.3.3 DSP Memory Transfers

Using DSP memory transfers, it is possible to access the memory space within the DSP. The memory spaces that can be accessed are shown below.

- All data memory
- Program memory (write only)
- Peripheral (circuitry) registers within the DSP (except for some parts)

On TWL, switching memory banks is the faster way of accessing the data memory and the program memory, but for DSP memory transfers, the memory being referenced by the DSP can be accessed simultaneously when the DSP is running.

Two 16-stage FIFOs have been prepared to make memory access more efficient during DSP memory transfers. One is used for reads, and the other for writes. The DSP configuration register (DSP_PCFG) can be used to notify the main processor of interrupts based on the FIFO status. Furthermore, the DSP status register (DSP_PSTS) can be used to determine the FIFO states.

The following two registers are used during DSP memory transfers.

DSP Transfer Data Register

Name	Address	Attribute	Initial Value
DSP_PDATA	0x04004300	R/W	0x0000

15	8	7	0
PDATA			

When the main processor reads this register, a value is popped from the read FIFO, and that value is returned.

When a value is written to this register during a write transfer, the value is pushed onto the write FIFO, and the DMA transfer will begin.

DSP Transfer Address Register

Name	Address	Attribute	Initial Value
DSP_PADR	0x04004304	W	0x0000

15	8	7	0
PADR			

Specifies the address of the DSP memory space to access. This register is used to specify the lower 16 bits. In order to specify the upper 16 bits, you must set a value in the DMA register within the DSP.

It is not possible to read values that have been written to this register. Reads will always return 0x0000.

22.3.4 Command / Reply Register

The system includes three sets each of 16-bit registers: three command registers (DSP_CMDx) for communication from the main processor to the DSP, and three reply registers (DSP_REPx) for communication from the DSP to the main processor.

The DSP core will be notified of writes to the command register by means of interrupts. It is possible to determine whether the DSP has read the command register by looking at the status bits (RCMDIM0-2) in the DSP status register (DSP_PSTS).

Depending on the settings in the DSP configuration register (DSP_PCFG), the main processor can be notified through an interrupt when the DSP writes to the reply register. The write can also be detected by polling the DSP status register without notification.

DSP Command Register 0-2 (ARM9 > DSP)

Name	Address	Attribute	Initial Value
DSP_CMDx (x = 0, 1, 2)	0x04004320, 0x04004328, 0x04004330	R/W	0x0000

15	8	7	0
APBP_CMDx			

Writes data from the main processor to be passed to the DSP.

It is possible to find out whether the DSP has read this register by looking up the corresponding bit in the DSP status register (RCMDIMx).

DSP Reply Register 0-2 (DSP > ARM9)

Name	Address	Attribute	Initial Value
DSP_REPx (x = 0, 1, 2)	0x04004324, 0x0400432C, 0x04004334	R	0x0000

15	8	7	0
APBP_REPx			

Reads data that is passed from the DSP to the main processor.

When the DSP updates this register, 1 will be set in the corresponding bit (PRIx) in the DSP status register. Also, if the corresponding bit (RRIEx) in the DSP configuration register is set to 1, the main processor is notified by an interrupt.

This update only supports writes to this register by the DSP, so even if the value is unchanged (that is, if the register is overwritten by the same value), it will be considered to have been updated.

22.3.5 Semaphore Register

The system includes a 16-bit (1 bit x 16) semaphore register, which uses interrupts, for mutual communication between the main processor and the DSP.

DSP Semaphore Register (ARM9 > DSP)

Name	Address	Attribute	Initial Value
DSP_PSEM	0x04004310	R/W	0x0000

15	8	7	0
PSEM			

Writing a 1 for a given bit will notify the DSP of an interrupt as long as that bit is not masked in the DSP's semaphore mask register (APBP_MASK).

DSP Semaphore Mask Register (DSP > ARM9)

Name	Address	Attribute	Initial Value
DSP_PMASK	0x04004314	R/W	0x0000

15	8	7	0
PMASK			

If a given bit is set to 1, the main processor will not be notified of an interrupt, even if a 1 is written to the corresponding bit of the DSP's semaphore register (APBP_PSEM).

DSP Semaphore Clear Register (DSP > ARM9)

Name	Address	Attribute	Initial Value
DSP_PCLEAR	0x04004318	W	0x0000

15	8	7	0
PCLEAR			

If 1 is written for a bit, the corresponding bit in the semaphore register (DSP_PSEM) will be cleared to 0. The content that was written will not be retained, so it is not necessary to write 0 after the bit is cleared.

DSP Semaphore Data Register (DSP > ARM9)

Name	Address	Attribute	Initial Value
DSP_SEM	0x0400431C	R	0x0000

15	8	7	0
APBP_PSEM			

It is possible to read the contents of the DSP's semaphore register (APBP_PSEM). Unless all the bits in the semaphore mask register (DSP_PMASK) that correspond to the 1 bits written by the DSP core are all set to 1, the main processor will be notified of an interrupt.

22.4 Procedures for Starting the DSP

To start up the DSP block, follow the procedure below.

1. Preparing the program and data

   Write the program for the DSP to the WRAM-B bank in the work RAM, write the data for the DSP to WRAM-C, and assign each memory block to the DSP.

2. Preparing to start by configuring the DSP's clock and reset configuration
   - i. Ensure that the DSP_RSTB bit in the system configuration’s SCFG_RST register is 0.
   - ii. Supply a clock to the DSP block by setting the DSP_HCLK bit in the system configuration's SCFG_CLK register to 1.
   - iii. After supplying the clock, wait 8 cycles on the DSP clock (134 MHz), then set the DSP_RSTB bit of the SCFG_RST register to 1 to cancel the reset on the DSP block.
   - iv. At this point, the DSP core will start, but the hardware logic will put it into the sleep state.

3. Boot Process
   - i. Write a 1 to the DSPR bit in the DSP configuration register (DSP_PCFG).
   - ii. Wait for the PRST bit in the DSP status register (DSP_PSTS) to become 0. (During this time, you can switch between the WRAM-B and WRAM-C banks and change their contents.)
   - iii. Write a 0 to the DSPR bit in the DSP configuration register.
   - iv. Execution of the program will begin starting with address 0 in the memory for DSP programs (the first address in the memory block for which the WRAM-B master is set to the DSP and the offset is set to 0).

To use the DSP block for the first time after the system reset, perform step 1 and 2 (it doesn't matter which step is performed first), then perform step 3. For the second and subsequent times the DSP is used, the DSP block can be initialized (with the exception of the DSP interface register) just by performing step 3.

To initialize the DSP interface register, follow the procedure shown below after getting to step 3-ii in the procedures above.

1. Write 0s to all the PRIE0-2 bits in the DSP configuration register.
2. Write 0x0000 to the DSP semaphore register (DSP_PSEM).
3. Write 0xFFFF to the DSP semaphore clear register (DSP_PCLEAR).
4. Perform a dummy read of DSP reply registers 0-2 (DSP_REPx), one by one.

23 NAND Flash Memory

The TWL includes NAND flash memory for saving images and downloaded applications.

The TWL's built-in NAND flash memory has the following characteristics.

- Capacity is 256 MB
- Read speeds vary between 0.8 and 4.0 MB/s
- Write speeds vary between 0.3 and 2.4 MB/s

There is comparatively little variation in read speeds over multiple reads; the variation in write speeds over multiple writes is comparatively high.

NAND flash memory degrades with repeated reads and writes, and the greater the number of accesses, the more often latency will occur (the latency can be as much as 13 milliseconds per 512-byte chunk of data). This has no impact on write speed.

Note: On rare occasions, the SDK will insert waits of 2 to 13 milliseconds regardless of whether NAND flash memory degradation exists.

Furthermore, the performance of both reads and writes will be markedly reduced if the file pointers are not 4-byte aligned.

NAND flash memory and SD Memory Cards cannot be accessed at the same time because these two components share the same interface.

When accessing NAND flash memory, be sure to use the API that has been provided in the SDK.

24 SD Memory Cards

The TWL comes with one SD Memory Card slot. The TWL's SD Memory Card slot has the following features:

- The supported media are (1) SD Memory Cards and (2) SDHC Memory Cards with capacities between 2 and 32 GB.
- SPI mode, encrypted authentication (CPRM / Content Protection for Recordable Media), and high-speed mode are **not supported**. (Even if high-speed memory card is used, it is not possible to access the cards faster than the speeds shown below.)
- Maximum read speed: 6.0 MB/s
- Maximum write speed: 2.4 MB/s

Images and NAND applications can be backed up on SD Memory Cards. The file system supports the SD standard format.

In addition to the standard access, applications can perform operations using insertion and removal of an SD Memory Card as a trigger. Applications can also look up the write-protection state of a card.

The access speeds depend not only on the host's control, but also on the control of the SD Card's internal controller. As a result, the access speeds will vary from one SD Memory Card to another.

miniSD and microSD cards can be accessed by using an SD Card adapter. However, inserting a microSD card into a miniSD adapter, and then inserting that into an SD Card adapter is prohibited, and data in such "doubly nested" cards may not always be accessed correctly.

Read and write performance will both be markedly reduced if the file pointers are not 4-byte aligned.

NAND flash memory and SD Memory Cards cannot be accessed at the same time because these two components share the same interface.

When accessing SD Memory Cards that have been inserted into the slot, be sure to use the API provided in the SDK.

Appendix A. Register List

A.1 Addresses 0x04000000 and Higher

Address[a] Offset	ARM9 Register Name	Page	Description
0X000	DISPCNT	79	2D Graphics Engine A display control
0X002			
0X004	DISPSTAT	75	Display status
0X006	VCOUNT	77	V count comparison
0X008	BG0CNT	105	2D Graphics Engine A BG0 control
0X00A	BG1CNT	105	2D Graphics Engine A BG1 control
0X00C	BG2CNT	107	2D Graphics Engine A BG2 control
0X00E	BG3CNT	107	2D Graphics Engine A BG3 control
0X010	BG0HOFS	129	2D Graphics Engine A BG0 display H offset
0X012	BG0VOFS	129	2D Graphics Engine A BG0 display V offset
0X014	BG1HOFS	129	2D Graphics Engine A BG1 display H offset
0X016	BG1VOFS	129	2D Graphics Engine A BG1 display V offset
0X018	BG2HOFS	129	2D Graphics Engine A BG2 display H offset
0X01A	BG2VOFS	129	2D Graphics Engine A BG2 display V offset
0X01C	BG3HOFS	129	2D Graphics Engine A BG3 display H offset
0X01E	BG3VOFS	129	2D Graphics Engine A BG3 display V offset
0X020	BG2PA	132	2D Graphics Engine A BG2 affine transformation parameters (same line X-direction reference shift dx)
0X022	BG2PB	132	2D Graphics Engine A BG2 affine transformation parameters (next line X-direction reference shift dmx)
0X024	BG2PC	132	2D Graphics Engine A BG2 affine transformation parameters (same line Y-direction reference shift dy)
0X026	BG2PD	132	2D Graphics Engine A BG2 affine transformation parameters (next line Y-direction reference shift dmy)
0X028	BG2X	131	2D Graphics Engine A BG2 reference start point (x coordinate)
0X02A			
0X02C	BG2Y	131	2D Graphics Engine A BG2 reference start point (y coordinate)
0X02E			
0X030	BG3PA	132	2D Graphics Engine A BG3 affine transformation parameters (same line X-direction reference shift dx)
0X032	BG3PB	132	2D Graphics Engine A BG3 affine transformation parameters (next line X-direction reference shift dmx)
0X034	BG3PC	132	2D Graphics Engine A BG3 affine transformation parameters (same line Y-direction reference shift dy)
0X036	BG3PD	132	2D Graphics Engine A BG3 affine transformation parameters (next line Y-direction reference shift dmy)
0X038	BG3X	131	2D Graphics Engine A BG3 reference start point (x coordinate)
0X03A			
0X03C	BG3Y	131	2D Graphics Engine A BG3 reference start point (y coordinate)
0X03E			
0X040	WIN0H	167	2D Graphics Engine A window 0 H size

Address[a] Offset	ARM9 Register Name	Page	Description
0X042	WIN1H	167	2D Graphics Engine A window 1 H size
0X044	WIN0V	167	2D Graphics Engine A window 0 V size
0X046	WIN1V	167	2D Graphics Engine A window 1 V size
0X048	WININ	166	2D Graphics Engine A window inside
0X04A	WINOUT	166	2D Graphics Engine A window outside
0X04C	MOSAIC	174	2D Graphics Engine A mosaic size
0X04E			
0X050	BLDCNT	170	2D Graphics Engine A color special effects
0X052	BLDALPHA	172	2D Graphics Engine A alpha blending factor
0X054	BLDY	173	2D Graphics Engine A brightness change factor
0X056			
0X058			
0X05A			
0X05C			
0X05E			
0X060	DISP3DCNT	179	3D display control
0X062			
0X064	DISPCAPCNT	91	Display capture
0X066			
0X068	DISP_MMEM_FIFO	89	Main memory display FIFO
0X06A			
0X06C	MASTER_BRIGHT	95	Image output A master brightness
0X06E			
0X070			
0X072			
0X074			
0X076			
0X078			
0X07A			
0X07C			
0X07E			
0X080			
0X082			
0X084			
0X086			
0X088			
0X08A			
0X08C			
0X08E			
0X090			
0X092			

Address[a] Offset	ARM9 Register Name	Page	Description
0X094			
0X096			
0X098			
0X09A			
0X09C			
0X09E			
0X0A0			
0X0A2			
0X0A4			
0X0A6			
0X0A8			
0X0AA			
0X0AC			
0X0AE			
0X0B0	DMA0SAD	297	DMA0 source address
0X0B2			
0X0B4	DMA0DAD	297	DMA0 destination address
0X0B6			
0X0B8	DMA0CNT	298	DMA0 control
0X0BA			
0X0BC	DMA1SAD	297	DMA1 source address
0X0BE			
0X0C0	DMA1DAD	297	DMA1 destination address
0X0C2			
0X0C4	DMA1CNT	298	DMA1 control
0X0C6			
0X0C8	DMA2SAD	297	DMA2 source address
0X0CA			
0X0CC	DMA2DAD	297	DMA2 destination address
0X0CE			
0X0D0	DMA2CNT	298	DMA2 control
0X0D2			
0X0D4	DMA3SAD	297	DMA3 source address
0X0D6			
0X0D8	DMA3DAD	297	DMA3 destination address
0X0DA			
0X0DC	DMA3CNT	298	DMA3 control
0X0DE			
0X0E0			
0X0E2			
0X0E4			

Address[a] Offset	ARM9 Register Name	Page	Description
0X0E6			
0X0E8			
0X0EA			
0X0EC			
0X0EE			
0X0F0			
0X0F2			
0X0F4			
0X0F6			
0X0F8			
0X0FA			
0X0FC			
0X0FE			
0X100	TM0CNT_L	309	Timer 0 counter
0X102	TM0CNT_H	309	Timer 0 control
0X104	TM1CNT_L	310	Timer 1 counter
0X106	TM1CNT_H	310	Timer 1 control
0X108	TM2CNT_L	310	Timer 2 counter
0X10A	TM2CNT_H	310	Timer 2 control
0X10C	TM3CNT_L	310	Timer 3 counter
0X10E	TM3CNT_H	310	Timer 3 control
0X110			
0X112			
0X114			
0X116			
0X118			
0X11A			
0X11C			
0X11E			
0X120			
0X122			
0X124			
0X126			
0X128			
0X12A			
0X12C			
0X12E			
0X130	KEYINPUT	329	Key input
0X132	KEYCNT	330	Key control
0X134			
0X136			

Address[a] Offset	ARM9 Register Name	Page	Description
0X138			
0X13A			
0X13C			
0X13E			
0X140			
0X142			
0X144			
0X146			
0X148			
0X14A			
0X14C			
0X14E			
0X150			
0X152			
0X154			
0X156			
0X158			
0X15A			
0X15C			
0X15E			
0X160			
0X162			
0X164			
0X166			
0X168			
0X16A			
0X16C			
0X16E			
0X170			
0X172			
0X174			
0X176			
0X178			
0X17A			
0X17C			
0X17E			
0X180			
0X182			
0X184			
0X186			
0X188			

Address[a] Offset	ARM9 Register Name	Page	Description
0X18A			
0X18C			
0X18E			
0X190			
0X192			
0X194			
0X196			
0X198			
0X19A			
0X19C			
0X19E			
0X1A0			
0X1A2			
0X1A4			
0X1A6			
0X1A8			
0X1AA			
0X1AC			
0X1AE			
0X1B0			
0X1B2			
0X1B4			
0X1B6			
0X1B8			
0X1BA			
0X1BC			
0X1BE			
0X1C0			
0X1C2			
0X1C4			
0X1C6			
0X1C8			
0X1CA			
0X1CC			
0X1CE			
0X1D0			
0X1D2			
0X1D4			
0X1D6			
0X1D8			
0X1DA			

Address[a] Offset	ARM9 Register Name	Page	Description
0X1DC			
0X1DE			
0X1E0			
0X1E2			
0X1E4			
0X1E6			
0X1E8			
0X1EA			
0X1EC			
0X1EE			
0X1F0			
0X1F2			
0X1F4			
0X1F6			
0X1F8			
0X1FA			
0X1FC			
0X1FE			
0X200			
0X202			
0X204	EXMEMCNT	20	External memory control
0X206			
0X208	IME	311	Interrupt master flag
0X20A			
0X20C			
0X20E			
0X210 0X212	IE	312	Interrupt enable flag
0X214 0X216	IF	314	Interrupt request flag
0X218			
0X21A			
0X21C			
0X21E			
0X220			
0X222			
0X224			
0X226			
0X228			
0X22A			
0X22C			

Address[a] Offset	ARM9 Register Name	Page	Description
0X22E			
0X230			
0X232			
0X234			
0X236			
0X238			
0X23A			
0X23C			
0X23E			
0X240	VRAMCNT	30	RAM bank control 0
0X242			
0X244	WVRAMCNT	33	RAM bank control 1
0X246			
0X248	VRAM_HI_CNT	36	RAM bank control 2
0X24A			
0X24C			
0X24E			
0X250			
0X252			
0X254			
0X256			
0X258			
0X25A			
0X25C			
0X25E			
0X260			
0X262			
0X264			
0X266			
0X268			
0X26A			
0X26C			
0X26E			
0X270			
0X272			
0X274			
0X276			
0X278			
0X27A			
0X27C			
0X27E			

Address[a] Offset	ARM9 Register Name	Page	Description
0X280	DIVCNT	323	Divider control
0X282			
0X284			
0X286			
0X288			
0X28A			
0X28C			
0X28E			
0X290	DIV_NUMER	323	Numerator
0X292			
0X294			
0X296			
0X298	DIV_DENOM	323	Denominator
0X29A			
0X29C			
0X29E			
0X2A0	DIV_RESULT	323	Quotient
0X2A2			
0X2A4			
0X2A6			
0X2A8	DIVREM_RESULT	323	Remainder
0X2AA			
0X2AC			
0X2AE			
0X2B0	SQRTCNT	326	Square root unit control
0X2B2			
0X2B4	SQRT_RESULT	326	Square root unit result
0X2B6			
0X2B8	SQRT_PARAM	326	Square root unit data
0X2BA			
0X2BC			
0X2BE			
0X2C0			
0X2C2			
0X2C4			
0X2C6			
0X2C8			
0X2CA			
0X2CC			
0X2CE			
0X2D0			

Address[a] Offset	ARM9 Register Name	Page	Description
0X2D2			
0X2D4			
0X2D6			
0X2D8			
0X2DA			
0X2DC			
0X2DE			
0X2E0			
0X2E2			
0X2E4			
0X2E6			
0X2E8			
0X2EA			
0X2EC			
0X2EE			
0X2F0			
0X2F2			
0X2F4			
0X2F6			
0X2F8			
0X2FA			
0X2FC			
0X2FE			
0X300			
0X302			
0X304	POWCNT	78	Power control
0X306			
0X308			
0X30A			
0X30C			
0X30E			
0X310			
0X312			
0X314			
0X316			
0X318			
0X31A			
0X31C			
0X31E			
0X320	RDLINES_COUNT	292	Rendering minimum fill
0X322			

Address[a] Offset	ARM9 Register Name	Page	Description
0X324			
0X326			
0X328			
0X32A			
0X32C			
0X32E			
0X330	EDGE_COLOR_0_L	284	Edge marking color (Polygon ID's 3 upper bits are 000)
0X332	EDGE_COLOR_0_H	284	Edge marking color (Polygon ID's 3 upper bits are 001)
0X334	EDGE_COLOR_1_L	284	Edge marking color (Polygon ID's 3 upper bits are 010)
0X336	EDGE_COLOR_1_H	284	Edge marking color (Polygon ID's 3 upper bits are 011)
0X338	EDGE_COLOR_2_L	284	Edge marking color (Polygon ID's 3 upper bits are 100)
0X33A	EDGE_COLOR_2_H	284	Edge marking color (Polygon ID's 3 upper bits are 101)
0X33C	EDGE_COLOR_3_L	284	Edge marking color (Polygon ID's 3 upper bits are 110)
0X33E	EDGE_COLOR_3_H	284	Edge marking color (Polygon ID's 3 upper bits are 111)
0X340	ALPHA_TEST_REF	283	Alpha test
0X342			
0X344			
0X346			
0X348			
0X34A			
0X34C			
0X34E			
0X350 0X352	CLEAR_COLOR	256	Color buffer initial value
0X354	CLEAR_DEptH	256	Depth buffer initial value
0X356	CLRIMAGE_OFFSET	258	Clear image offset
0X358 0X35A	FOG_COLOR	285	Fog color
0X35C	FOG_OFFSET	285	Fog offset
0X35E			
0X360	FOG_TABLE_0_L	286	Fog density table (0, 1)
0X362	FOG_TABLE_0_H	286	Fog density table (2, 3)
0X364	FOG_TABLE_1_L	286	Fog density table (4, 5)
0X366	FOG_TABLE_1_H	286	Fog density table (6, 7)

Address[a] Offset	ARM9 Register Name	Page	Description
0X368	FOG_TABLE_2_L	286	Fog density table (8, 9)
0X36A	FOG_TABLE_2_H	286	Fog density table (10, 11)
0X36C	FOG_TABLE_3_L	286	Fog density table (12, 13)
0X36E	FOG_TABLE_3_H	286	Fog density table (14, 15)
0X370	FOG_TABLE_4_L	286	Fog density table (16, 17)
0X372	FOG_TABLE_4_H	286	Fog density table (18, 19)
0X374	FOG_TABLE_5_L	286	Fog density table (20, 21)
0X376	FOG_TABLE_5_H	286	Fog density table (22, 23)
0X378	FOG_TABLE_6_L	286	Fog density table (24, 25)
0X37A	FOG_TABLE_6_H	286	Fog density table (26, 27)
0X37C	FOG_TABLE_7_L	286	Fog density table (28, 29)
0X37E	FOG_TABLE_7_H	286	Fog density table (30, 31)
0X380	TOON_TABLE_0_L	266	Toon table (RGB conversion value when brightness is 0)
0X382	TOON_TABLE_0_H	266	Toon table (RGB conversion value when brightness is 1)
0X384	TOON_TABLE_1_L	266	Toon table (RGB conversion value when brightness is 2)
0X386	TOON_TABLE_1_H	266	Toon table (RGB conversion value when brightness is 3)
0X388	TOON_TABLE_2_L	266	Toon table (RGB conversion value when brightness is 4)
0X38A	TOON_TABLE_2_H	266	Toon table (RGB conversion value when brightness is 5)
0X38C	TOON_TABLE_3_L	266	Toon table (RGB conversion value when brightness is 6)
0X38E	TOON_TABLE_3_H	266	Toon table (RGB conversion value when brightness is 7)
0X390	TOON_TABLE_4_L	266	Toon table (RGB conversion value when brightness is 8)
0X392	TOON_TABLE_4_H	266	Toon table (RGB conversion value when brightness is 9)
0X394	TOON_TABLE_5_L	266	Toon table (RGB conversion value when brightness is 10)
0X396	TOON_TABLE_5_H	266	Toon table (RGB conversion value when brightness is 11)
0X398	TOON_TABLE_6_L	266	Toon table (RGB conversion value when brightness is 12)
0X39A	TOON_TABLE_6_H	266	Toon table (RGB conversion value when brightness is 13)
0X39C	TOON_TABLE_7_L	266	Toon table (RGB conversion value when brightness is 14)
0X39E	TOON_TABLE_7_H	266	Toon table (RGB conversion value when brightness is 15)
0X3A0	TOON_TABLE_8_L	266	Toon table (RGB conversion value when brightness is 16)
0X3A2	TOON_TABLE_8_H	266	Toon table (RGB conversion value when brightness is 17)
0X3A4	TOON_TABLE_9_L	266	Toon table (RGB conversion value when brightness is 18)
0X3A6	TOON_TABLE_9_H	266	Toon table (RGB conversion value when brightness is 19)
0X3A8	TOON_TABLE_10_L	266	Toon table (RGB conversion value when brightness is 20)
0X3AA	TOON_TABLE_10_H	266	Toon table (RGB conversion value when brightness is 21)
0X3AC	TOON_TABLE_11_L	266	Toon table (RGB conversion value when brightness is 22)
0X3AE	TOON_TABLE_11_H	266	Toon table (RGB conversion value when brightness is 23)
0X3B0	TOON_TABLE_12_L	266	Toon table (RGB conversion value when brightness is 24)
0X3B2	TOON_TABLE_12_H	266	Toon table (RGB conversion value when brightness is 25)
0X3B4	TOON_TABLE_13_L	266	Toon table (RGB conversion value when brightness is 26)
0X3B6	TOON_TABLE_13_H	266	Toon table (RGB conversion value when brightness is 27)
0X3B8	TOON_TABLE_14_L	266	Toon table (RGB conversion value when brightness is 28)

Address[a] Offset	ARM9 Register Name	Page	Description
0X3BA	TOON_TABLE_14_H	266	Toon table (RGB conversion value when brightness is 29)
0X3BC	TOON_TABLE_15_L	266	Toon table (RGB conversion value when brightness is 30)
0X3BE	TOON_TABLE_15_H	266	Toon table (RGB conversion value when brightness is 31)
0X3C0			
0X3C2			
0X3C4			
0X3C6			
0X3C8			
0X3CA			
0X3CC			
0X3CE			
0X3D0			
0X3D2			
0X3D4			
0X3D6			
0X3D8			
0X3DA			
0X3DC			
0X3DE			
0X3E0			
0X3E2			
0X3E4			
0X3E6			
0X3E8			
0X3EA			
0X3EC			
0X3EE			
0X3F0			
0X3F2			
0X3F4			
0X3F6			
0X3F8			
0X3FA			
0X3FC			
0X3FE			
0X400	GXFIFO	192	Geometry command FIFO
0X402			

Address[a] Offset	ARM9 Register Name	Page	Description
0X404	GXFIFO image		
0X406			
0X408			
0X40A			
0X40C			
0X40E			
0X410			
0X412			
0X414			
0X416			
0X418			
0X41A			
0X41C			
0X41E			
0X420			
0X422			
0X424			
0X426			
0X428			
0X42A			
0X42C			
0X42E			
0X430			
0X432			
0X434			
0X436			
0X438			
0X43A			
0X43C			
0X43E			
0X440	MTX_MODE	206	Current matrix mode setting
0X442			
0X444	MTX_PUSH	211	Push current matrix onto the stack
0X446			
0X448	MTX_POP	211	Pop current matrix from the stack
0X44A			
0X44C	MTX_STORE	212	Store current matrix in specified position in the stack
0X44E			
0X450	MTX_RESTORE	212	Read matrix from specified position in the stack
0X452			

Address[a] Offset	ARM9 Register Name	Page	Description
0X454 0X456	MTX_IDENTITY	207	Initialize unit matrix
0X458 0X45A	MTX_LOAD_4x4	207	Set 4x4 matrix
0X45C 0X45E	MTX_LOAD_4x3	207	Set 4x3 matrix
0X460 0X462	MTX_MULT_4x4	208	Multiply by 4x4 matrix
0X464 0X466	MTX_MULT_4x3	208	Multiply by 4x3 matrix
0X468 0X46A	MTX_MULT_3x3	209	Multiply by 3x3 matrix
0X46C 0X46E	MTX_SCALE	210	Multiply by the Scale matrix
0X470 0X472	MTX_TRANS	209	Multiply by the Translation matrix
0X474			
0X476			
0X478			
0X47A			
0X47C			
0X47E			
0X480 0X482	COLOR	226	Vertex color
0X484 0X486	NORMAL	227	Normal vector
0X488 0X48A	TEXCOORD	231	Texture coordinates
0X48C 0X48E	VTX_16	227	Vertex coordinates
0X490 0X492	VTX_10	228	Vertex coordinates
0X494 0X496	VTX_XY	228	Vertex XY coordinates
0X498 0X49A	VTX_XZ	228	Vertex XZ coordinates
0X49C 0X49E	VTX_YZ	228	Vertex YZ coordinates
0X4A0 0X4A2	VTX_DIFF	229	Vertex coordinates difference value specification

Address[a] Offset	ARM9 Register Name	Page	Description
0X4A4	POLYGON_ATTR	221	Polygon-related attribute values
0X4A6			
0X4A8	TEXIMAGE_PARAM	232	Texture parameters
0X4AA			
0X4AC	TEXPLTT_BASE	237	Texture palette base address
0X4AE			
0X4B0			
0X4B2			
0X4B4			
0X4B6			
0X4B8			
0X4BA			
0X4BC			
0X4BE			
0X4C0	DIF_AMB	217	Material's diffuse and ambient colors
0X4C2			
0X4C4	SPE_EMI	217	Material's specular reflection and emitted light colors
0X4C6			
0X4C8	LIGHT_VECTOR	214	Light direction vector
0X4CA			
0X4CC	LIGHT_COLOR	214	Light color
0X4CE			
0X4D0	SHININESS	218	Specular reflection shininess table
0X4D2			
0X4D4			
0X4D6			
0X4D8			
0X4DA			
0X4DC			
0X4DE			
0X4E0			
0X4E2			
0X4E4			
0X4E6			
0X4E8			
0X4EA			
0X4EC			
0X4EE			
0X4F0			
0X4F2			
0X4F4			

Address[a] Offset	ARM9 Register Name	Page	Description
0X4F6			
0X4F8			
0X4FA			
0X4FC			
0X4FE			
0X500	BEGIN_VTXS	225	Vertex list start
0X502			
0X504	END_VTXS	226	Vertex list end
0X506			
0X508			
0X50A			
0X50C			
0X50E			
0X510			
0X512			
0X514			
0X516			
0X518			
0X51A			
0X51C			
0X51E			
0X520			
0X522			
0X524			
0X526			
0X528			
0X52A			
0X52C			
0X52E			
0X530			
0X532			
0X534			
0X536			
0X538			
0X53A			
0X53C			
0X53E			
0X540	SWAP_BUFFERS	203	Swap data group
0X542			
0X544			
0X546			

Address[a] Offset	ARM9 Register Name	Page	Description
0X548			
0X54A			
0X54C			
0X54E			
0X550			
0X552			
0X554			
0X556			
0X558			
0X55A			
0X55C			
0X55E			
0X560			
0X562			
0X564			
0X566			
0X568			
0X56A			
0X56C			
0X56E			
0X570			
0X572			
0X574			
0X576			
0X578			
0X57A			
0X57C			
0X57E			
0X580	VIEWPORT	205	Viewport
0X582			
0X584			
0X586			
0X588			
0X58A			
0X58C			
0X58E			
0X590			
0X592			
0X594			
0X596			
0X598			

Address[a] Offset	ARM9 Register Name	Page	Description
0X59A			
0X59C			
0X59E			
0X5A0			
0X5A2			
0X5A4			
0X5A6			
0X5A8			
0X5AA			
0X5AC			
0X5AE			
0X5B0			
0X5B2			
0X5B4			
0X5B6			
0X5B8			
0X5BA			
0X5BC			
0X5BE			
0X5C0 0X5C2	BOX_TEST	241	Box test
0X5C4 0X5C6	POS_TEST	243	Position coordinate test
0X5C8 0X5CA	VEC_TEST	243	Direction vector test
0X5CC			
0X5CE			
0X5D0			
0X5D2			
0X5D4			
0X5D6			
0X5D8			
0X5DA			
0X5DC			
0X5DE			
0X5E0			
0X5E2			
0X5E4			
0X5E6			
0X5E8			
0X5EA			

Address[a] Offset	ARM9 Register Name	Page	Description
0X5EC			
0X5EE			
0X5F0			
0X5F2			
0X5F4			
0X5F6			
0X5F8			
0X5FA			
0X5FC			
0X5FE			
0X600	GXSTAT	245	Geometry engine status
0X602			
0X604	LISTRAM_COUNT	247	Polygon list RAM count
0X606	VTXRAM_COUNT	247	Vertex RAM count
0X608			
0X60A			
0X60C			
0X60E			
0X610	DISP_1DOT_DEptH	224	1-dot polygon display boundary depth value
0X612			
0X614			
0X616			
0X618			
0X61A			
0X61C			
0X61E			
0X620	POS_RESULT_X	243	Result of position coordinate test (clip coordinate X component)
0X622			
0X624	POS_RESULT_Y	243	Result of position coordinate test (clip coordinate Y component)
0X626			
0X628	POS_RESULT_Z	243	Result of position coordinate test (clip coordinate Z component)
0X62A			
0X62C	POS_RESULT_W	243	Result of position coordinate test (clip coordinate W component)
0X62E			
0X630	VEC_RESULT_X	244	Result of direction vector test (X component)
0X632	VEC_RESULT_Y	244	Result of direction vector test (Y component)
0X634	VEC_RESULT_Z	244	Result of direction vector test (Z component)
0X636			
0X638			
0X63A			
0X63C			

Address[a] Offset	ARM9 Register Name	Page	Description
0X63E			
0X640	CLIPMTX_RESULT_0	213	Current clip coordinate matrix (element m0)
0X642			
0X644	CLIPMTX_RESULT_1	213	Current clip coordinate matrix (element m1)
0X646			
0X648	CLIPMTX_RESULT_2	213	Current clip coordinate matrix (element m2)
0X64A			
0X64C	CLIPMTX_RESULT_3	213	Current clip coordinate matrix (element m3)
0X64E			
0X650	CLIPMTX_RESULT_4	213	Current clip coordinate matrix (element m4)
0X652			
0X654	CLIPMTX_RESULT_5	213	Current clip coordinate matrix (element m5)
0X656			
0X658	CLIPMTX_RESULT_6	213	Current clip coordinate matrix (element m6)
0X65A			
0X65C	CLIPMTX_RESULT_7	213	Current clip coordinate matrix (element m7)
0X65E			
0X660	CLIPMTX_RESULT_8	213	Current clip coordinate matrix (element m8)
0X662			
0X664	CLIPMTX_RESULT_9	213	Current clip coordinate matrix (element m9)
0X666			
0X668	CLIPMTX_RESULT_10	213	Current clip coordinate matrix (element m10)
0X66A			
0X66C	CLIPMTX_RESULT_11	213	Current clip coordinate matrix (element m11)
0X66E			
0X670	CLIPMTX_RESULT_12	213	Current clip coordinate matrix (element m12)
0X672			
0X674	CLIPMTX_RESULT_13	213	Current clip coordinate matrix (element m13)
0X676			
0X678	CLIPMTX_RESULT_14	213	Current clip coordinate matrix (element m14)
0X67A			
0X67C	CLIPMTX_RESULT_15	213	Current clip coordinate matrix (element m15)
0X67E			
0X680	VECMTX_RESULT_0	213	Current direction vector matrix (element m0)
0X682			
0X684	VECMTX_RESULT_1	213	Current direction vector matrix (element m1)
0X686			
0X688	VECMTX_RESULT_2	213	Current direction vector matrix (element m2)
0X68A			
0X68C	VECMTX_RESULT_3	213	Current direction vector matrix (element m3)
0X68E			

Address[a] Offset	ARM9 Register Name	Page	Description
0X690	VECMTX_RESULT_4	213	Current direction vector matrix (element m4)
0X692			
0X694	VECMTX_RESULT_5	213	Current direction vector matrix (element m5)
0X696			
0X698	VECMTX_RESULT_6	213	Current direction vector matrix (element m6)
0X69A			
0X69C	VECMTX_RESULT_7	213	Current direction vector matrix (element m7)
0X69E			
0X6A0	VECMTX_RESULT_8	213	Current direction vector matrix (element m8)
0X6A2			
0X6A4			
0X6A6			
0X6A8			
0X6AA			
0X6AC			
0X6AE			
0X6B0			
0X6B2			
0X6B4			
0X6B6			
0X6B8			
0X6BA			
0X6BC			
0X6BE			
0X6C0			
0X6C2			
0X6C4			
0X6C6			
0X6C8			
0X6CA			
0X6CC			
0X6CE			
0X6D0			
0X6D2			
0X6D4			
0X6D6			
0X6D8			
0X6DA			
0X6DC			
0X6DE			
0X6E0			

Address[a] Offset	ARM9 Register Name	Page	Description
0X6E2			
0X6E4			
0X6E6			
0X6E8			
0X6EA			
0X6EC			
0X6EE			
0X6F0			
0X6F2			
0X6F4			
0X6F6			
0X6F8			
0X6FA			
0X6FC			
0X6FE			

a. This indicates the offset value from 0x04000000.

A.2 Addresses 0x04001000 and Higher (2D Graphics Engine B-related)

Address Offset	ARM9 Register Name	Page	Explanation
0X000	DB_DISPCNT	81	2D Graphics Engine B display control
0X002			
0X004			
0X006			
0X008	DB_BG0CNT	105	2D Graphics Engine B BG0 control
0X00A	DB_BG1CNT	105	2D Graphics Engine B BG1 control
0X00C	DB_BG2CNT	107	2D Graphics Engine B BG2 control
0X00E	DB_BG3CNT	107	2D Graphics Engine B BG3 control
0X010	DB_BG0HOFS	129	2D Graphics Engine B BG0 display H offset
0X012	DB_BG0VOFS	129	2D Graphics Engine B BG0 display V offset
0X014	DB_BG1HOFS	129	2D Graphics Engine B BG1 display H offset
0X016	DB_BG1VOFS	129	2D Graphics Engine B BG1 display V offset
0X018	DB_BG2HOFS	129	2D Graphics Engine B BG2 display H offset
0X01A	DB_BG2VOFS	129	2D Graphics Engine B BG2 display V offset
0X01C	DB_BG3HOFS	129	2D Graphics Engine B BG3 display H offset
0X01E	DB_BG3VOFS	129	2D Graphics Engine B BG3 display V offset
0X020	DB_BG2PA	132	2D Graphics Engine B BG2 affine transformation parameters (same line X-direction reference shift dx)
0X022	DB_BG2PB	132	2D Graphics Engine B BG2 affine transformation parameters (next line X-direction reference shift dmx)
0X024	DB_BG2PC	132	2D Graphics Engine B BG2 affine transformation parameters (same line Y-direction reference shift dy)
0X026	DB_BG2PD	132	2D Graphics Engine B BG2 affine transformation parameters (next line Y-direction reference shift dmy)
0X028	DB_BG2X	131	2D Graphics Engine B BG2 reference start point (x coordinate)
0X02A			
0X02C	DB_BG2Y	131	2D Graphics Engine B BG2 reference start point (y coordinate)
0X02E			
0X030	DB_BG3PA	132	2D Graphics Engine B BG3 affine transformation parameters (same line X-direction reference shift dx)
0X032	DB_BG3PB	132	2D Graphics Engine B BG3 affine transformation parameters (next line X-direction reference shift dmx)
0X034	DB_BG3PC	132	2D Graphics Engine B BG3 affine transformation parameters (same line Y-direction reference shift dy)
0X036	DB_BG3PD	132	2D Graphics Engine B BG3 affine transformation parameters (next line Y-direction reference shift dmy)
0X038	DB_BG3X	131	2D Graphics Engine B BG3 reference start point (x coordinate)
0X03A			

Address Offset	ARM9 Register Name	Page	Explanation
0X03C	DB_BG3Y	131	2D Graphics Engine B BG3 reference start point (y coordinate)
0X03E			
0X040	DB_WIN0H	167	2D Graphics Engine B window 0H size
0X042	DB_WIN1H	167	2D Graphics Engine B window 1H size
0X044	DB_WIN0V	167	2D Graphics Engine B window 0V size
0X046	DB_WIN1V	167	2D Graphics Engine B window 1V size
0X048	DB_WININ	166	2D Graphics Engine B window inside
0X04A	DB_WINOUT	166	2D Graphics Engine B window outside
0X04C	DB_MOSAIC	174	2D Graphics Engine B mosaic size
0X04E			
0X050	DB_BLDCNT	170	2D Graphics Engine B color special effects
0X052	DB_BLDALPHA	172	2D Graphics Engine B alpha blending factor
0X054	DB_BLDY	173	2D Graphics Engine B brightness conversion factor
0X056			
0X058			
0X05A			
0X05C			
0X05E			
0X060			
0X062			
0X064			
0X066			
0X068			
0X06A			
0X06C	DB_MASTER_BRIGHT	95	Image output B master brightness

A.3 Addresses 0x04004000 and Higher (TWL Extension-Related)

Address Offset	ARM9 Register Name	Page	Explanation
0X000	SCFG_A9ROM	9	ROM Status
0X002			
0X004	SCFG_CLK	10	New Block Clock Control
0X006	SCFG_RST	11	New Block Reset Control
0X008	SCFG_EXT	12	Extended Features Control
0X00A			
0X00C			
0X00E			
0X010	SCFG_MC		Memory Card Interface Status
0X012			
0X014			
0X016			
0X018			
0X01A			
0X01C			
0X01E			
0X020			
0X022			
0X024			
0X026			
0X028			
0X02A			
0X02C			
0X02E			
0X030			
0X032			
0X034			
0X036			
0X038			
0X03A			
0X03C			
0X03E			
0X040	MBK1	42	WRAM Bank Control Register 1
0X042			
0X044	MBK2	45	WRAM Bank Control Register 2
0X046			

Address Offset	ARM9 Register Name	Page	Explanation
0X048	MBK3	46	WRAM Bank Control Register 3
0X04A			
0X04C	MBK4	49	WRAM Bank Control Register 4
0X04E			
0X050	MBK5	50	WRAM Bank Control Register 5
0X052			
0X054	MBK6	43	WRAM Bank Control Register 6
0X056			
0X058	MBK7	47	WRAM Bank Control Register 7
0X05A			
0X05C	MBK8	52	WRAM Bank Control Register 8
0X05E			
0X060	MBK9		WRAM Bank Control Register 9
0X062			
0X064			
0X066			
0X068			
0X06A			
0X06C			
0X06E			
0X070			
0X072			
0X074			
0X076			
0X078			
0X07A			
0X07C			
0X07E			
0X080			
0X082			
0X084			
0X086			
0X088			
0X08A			
0X08C			
0X08E			
0X090			

Address Offset	ARM9 Register Name	Page	Explanation
0X092			
0X094			
0X096			
0X098			
0X09A			
0X09C			
0X09E			
0X0A0			
0X0A2			
0X0A4			
0X0A6			
0X0A8			
0X0AA			
0X0AC			
0X0AE			
0X0B0			
0X0B2			
0X0B4			
0X0B6			
0X0B8			
0X0BA			
0X0BC			
0X0BE			
0X0C0			
0X0C2			
0X0C4			
0X0C6			
0X0C8			
0X0CA			
0X0CC			
0X0CE			
0X0D0			
0X0D2			
0X0D4			
0X0D6			
0X0D8			
0X0DA			
0X0DC			

Address Offset	ARM9 Register Name	Page	Explanation
0X0DE			
0X0E0			
0X0E2			
0X0E4			
0X0E6			
0X0E8			
0X0EA			
0X0EC			
0X0EE			
0X0F0			
0X0F2			
0X0F4			
0X0F6			
0X0F8			
0X0FA			
0X0FC			
0X0FE			
0X100 0X102	NDMAGCNT	302	NDMA Global Control
0X104 0X106	NDMA0SAD	303	NDMA0 Source Address
0X108 0X10A	NDMA0DAD	303	NDMA0 Destination Address
0X10C 0X10E	NDMA0TCNT	303	NDMA0 Total Words Transferred Count
0X110 0X112	NDMA0WCNT	304	NDMA0 Word Count
0X114 0X116	NDMA0BCNT	304	NDMA0 Block Transfer Interval
0X118 0X11A	NDMA0FDATA	304	NDMA0 Fill Data
0X11C 0X11E	NDMA0CNT	305	NDMA0 Control
0X120 0X122	NDMA1SAD	303	NDMA1 Source Address
0X124 0X126	NDMA1DAD	303	NDMA1 Destination Address

TWL-06-0017-001-D
Released: February 16, 2009

Address Offset	ARM9 Register Name	Page	Explanation
0X128 0X12A	NDMA1TCNT	303	NDMA1 Total Words Transferred Count
0X12C 0X12E	NDMA1WCNT	304	NDMA1 Word Count
0X130 0X132	NDMA1BCNT	304	NDMA1 Block Transfer Interval
0X134 0X136	NDMA1FDATA	304	NDMA1 Fill Data
0X138 0X13A	NDMA1CNT	305	NDMA1 Control
0X13C 0X13E	NDMA2SAD	303	NDMA2 Source Address
0X140 0X142	NDMA2DAD	303	NDMA2 Destination Address
0X144 0X146	NDMA2TCNT	303	NDMA2 Total Words Transferred Count
0X148 0X14A	NDMA2WCNT	304	NDMA2 Word Count
0X14C 0X14E	NDMA2BCNT	304	NDMA2 Block Transfer Interval
0X150 0X152	NDMA2FDATA	304	NDMA2 Fill Data
0X154 0X156	NDMA2CNT	305	NDMA2 Control
0X158 0X15A	NDMA3SAD	303	NDMA3 Source Address
0X15C 0X15E	NDMA3DAD	303	NDMA3 Destination Address
0X160 0X162	NDMA3TCNT	303	NDMA3 Total Words Transferred Count
0X164 0X166	NDMA3WCNT	304	NDMA3 Word Count
0X168 0X16A	NDMA3BCNT	304	NDMA3 Block Transfer Interval
0X16C 0X16E	NDMA3FDATA	304	NDMA3 Fill Data
0X170 0X172	NDMA3CNT	305	NDMA3 Control

Address Offset	ARM9 Register Name	Page	Explanation
0X174			
0X176			
0X178			
0X17A			
0X17C			
0X17E			
0X180			
0X182			
0X184			
0X186			
0X188			
0X18A			
0X18C			
0X18E			
0X190			
0X192			
0X194			
0X196			
0X198			
0X19A			
0X19C			
0X19E			
0X1A0			
0X1A2			
0X1A4			
0X1A6			
0X1A8			
0X1AA			
0X1AC			
0X1AE			
0X1B0			
0X1B2			
0X1B4			
0X1B6			
0X1B8			
0X1BA			
0X1BC			
0X1BE			

TWL-06-0017-001-D
Released: February 16, 2009

Address Offset	ARM9 Register Name	Page	Explanation
0X1C0			
0X1C2			
0X1C4			
0X1C6			
0X1C8			
0X1CA			
0X1CC			
0X1CE			
0X1D0			
0X1D2			
0X1D4			
0X1D6			
0X1D8			
0X1DA			
0X1DC			
0X1DE			
0X1E0			
0X1E2			
0X1E4			
0X1E6			
0X1E8			
0X1EA			
0X1EC			
0X1EE			
0X1F0			
0X1F2			
0X1F4			
0X1F6			
0X1F8			
0X1FA			
0X1FC			
0X1FE			
0X200	CAM_MCNT		Camera Module Control
0X202	CAM_CNT		Camera Control
0X204	CAM_DAT		Camera Data
0X206			
0X208			
0X20A			

Address Offset	ARM9 Register Name	Page	Explanation
0X20C			
0X20E			
0X210	CAM_SOFS		Camera Trimming Starting Position Setting
0X212			
0X214	CAM_EOFS		Camera Trimming Ending Position Setting
0X216			
0X218			
0X21A			
0X21C			
0X21E			
0X220			
0X222			
0X224			
0X226			
0X228			
0X22A			
0X22C			
0X22E			
0X230			
0X232			
0X234			
0X236			
0X238			
0X23A			
0X23C			
0X23E			
0X240			
0X242			
0X244			
0X246			
0X248			
0X24A			
0X24C			
0X24E			
0X250			
0X252			
0X254			
0X256			

Address Offset	ARM9 Register Name	Page	Explanation
0X258			
0X25A			
0X25C			
0X25E			
0X260			
0X262			
0X264			
0X266			
0X268			
0X26A			
0X26C			
0X26E			
0X270			
0X272			
0X274			
0X276			
0X278			
0X27A			
0X27C			
0X27E			
0X280			
0X282			
0X284			
0X286			
0X288			
0X28A			
0X28C			
0X28E			
0X290			
0X292			
0X294			
0X296			
0X298			
0X29A			
0X29C			
0X29E			
0X2A0			
0X2A2			

Address Offset	ARM9 Register Name	Page	Explanation
0X2A4			
0X2A6			
0X2A8			
0X2AA			
0X2AC			
0X2AE			
0X2B0			
0X2B2			
0X2B4			
0X2B6			
0X2B8			
0X2BA			
0X2BC			
0X2BE			
0X2C0			
0X2C2			
0X2C4			
0X2C6			
0X2C8			
0X2CA			
0X2CC			
0X2CE			
0X2D0			
0X2D2			
0X2D4			
0X2D6			
0X2D8			
0X2DA			
0X2DC			
0X2DE			
0X2E0			
0X2E2			
0X2E4			
0X2E6			
0X2E8			
0X2EA			
0X2EC			
0X2EE			

Address Offset	ARM9 Register Name	Page	Explanation
0X2F0			
0X2F2			
0X2F4			
0X2F6			
0X2F8			
0X2FA			
0X2FC			
0X2FE			
0X300	DSP_PDATA	375	DSP Transfer Data
0X302			
0X304	DSP_PADR	375	DSP Transfer Address
0X306			
0X308	DSP_PCFG	372	DSP Configuration
0X30A			
0X30C	DSP_PSTS	373	DSP Status
0X30E			
0X310	DSP_PSEM	376	DSP Semaphore
0X312			
0X314	DSP_PMASK	376	DSP Semaphore Mask
0X316			
0X318	DSP_PCLEAR	377	DSP Semaphore Clear
0X31A			
0X31C	DSP_SEM	377	DSP Semaphore Data
0X31E			
0X320	DSP_CMD0	375	DSP Command Register 0
0X322			
0X324	DSP_REP0	376	DSP Reply Register 0
0X326			
0X328	DSP_CMD1	375	DSP Command Register 1
0X32A			
0X32C	DSP_REP1	376	DSP Reply Register 1
0X32E			
0X330	DSP_CMD2	375	DSP Command Register 2
0X332			
0X334	DSP_REP2	376	DSP Reply Register 2
0X336			
0X338			
0X33A			

Address Offset	ARM9 Register Name	Page	Explanation
0X33C			
0X33E			
0X340			
0X342			
0X344			
0X346			
0X348			
0X34A			
0X34C			
0X34E			
0X350			
0X352			
0X354			
0X356			
0X358			
0X35A			
0X35C			
0X35E			
0X360			
0X362			
0X364			
0X366			
0X368			
0X36A			
0X36C			
0X36E			
0X370			
0X372			
0X374			
0X376			
0X378			
0X37A			
0X37C			
0X37E			

Appendix B. List of VRAM Data Capacities

(Data capacity unit: bytes)

Size Format	8 x 8	16 x 16	32 x 32	64 x 64	128 x 128	256 x 192	256 x 256	512 x 256	512 x 512	1024 x 512	1024 x 1024
16-Color Character	32	128	512	2K	X	X	X	X	X	X	X
256-Color Character	64	256	1K	4K	X	X	X	X	X	X	X
Direct Color Bitmap OBJ	128	512	2K	8K	X	X	X	X	X	X	X
Normal Character BG Screen	X	X	X	X	X	X	2K	4K	8K	X	X
Rotated Character BG Screen	X	X	X	X	256	X	1K	X	4K	X	16K
Extended/Rotated Character BG Screen	X	X	X	X	512	X	2K	X	8K	X	32K
256-Color Bitmap BG	X	X	X	X	16K	X	64K	128K	256K	X	X
Large Screen 256-Color Bitmap BG	X	X	X	X	X	X	X	X	X	512K	X
Direct Color Bitmap BG	X	X	X	X	32K	X	128K	256K	512K	X	X
Clear Color Image	X	X	X	X	X	96K	X	X	X	X	X
Clear Depth Image	X	X	X	X	X	96K	X	X	X	X	X
Display Capture	X	X	X	X	32K	96K	X	X	X	X	X
4-Color Texture	16	64	256	1K	4K	—	16K	—	64K	—	256K
16-Color Texture	32	128	512	2K	8K	—	32K	—	128K	—	512K
256-Color Texture	64	256	1K	4K	16K	—	64K	—	256K	512K	X
A3I5 Translucent Texture	64	256	1K	4K	16K	—	64K	—	256K	512K	X
A5I3 Translucent Texture	64	256	1K	4K	16K	—	64K	—	256K	512K	X
Direct Color Texture	128	512	2K	8K	32K	—	128K	—	512K	X	X
Compressed Texture Image	16	64	256	1K	4K	—	16K	—	64K	—	256K
Compressed Texture Index	8	32	128	512	2K	—	8K	—	32K	—	128K
Maximum No. of Compressed Texture Interpolation Palettes	16	64	256	1K	4K	—	16K	—	64K	(96K)	(96K)

X: Outside of specifications

—: Omitted

Bold: Maximum value

( ): The maximum data value exceeds the RAM capacity. Therefore the maximum usable RAM capacity is specified.

TWL-06-0017-001-D
Released: February 16, 2009

Appendix C. Data Formats

BG, OBJ Character Data

d Format	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
16-color Character	P3				P2				P1				P0			
256-color Character	P1								P0							

Bitmap OBJ

d Format	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct Color Bitmap OBJ	ALPHA	BLUE					GREEN					RED				

BG Screen Data

d Format	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Normal Character BG Screen	Palette				Vflip	Hflip	Character name									
Rotate Character BG Screen									Character name							
Expand/rotate Character BG Screen	Palette				Vflip	Hflip	Character name									

Bitmap BG Data

d Format	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
256-color Bitmap BG									Color No.							
Large Screen 256-color Bitmap BG									Color No.							
Direct Color Bitmap BG	ALPHA	BLUE					GREEN					RED				

Palette Data

d Format	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BG, OBJ Palette Color	GREEN	BLUE					GREEN					RED				

Note: The fifteenth bit is used as the lowest-order green bit. The lowest-order bits for blue and red are extended with a 0.

Other Graphics Function Data

<table>
<tr><th>Format \ d</th><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>Clear Color Image</td><td>ALPHA</td><td colspan="5">BLUE</td><td colspan="5">GREEN</td><td colspan="5">RED</td></tr>
<tr><td rowspan="2">Clear Depth Image</td><td rowspan="2">FOG</td><td colspan="12">Integer part</td><td colspan="3">Decimal part</td></tr>
<tr><td colspan="15">Clear depth</td></tr>
<tr><td>Display Capture</td><td>ALPHA</td><td colspan="5">BLUE</td><td colspan="5">GREEN</td><td colspan="5">RED</td></tr>
</table>

Texture Data

<table>
<tr><th>Format \ d</th><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>4-color Texture</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="2">T0</td></tr>
<tr><td>16-color Texture</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="4">T0</td></tr>
<tr><td>256-color Texture</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="8">T0</td></tr>
<tr><td>A5I3 Translucent Texture</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="5">ALPHA</td><td colspan="3">INDEX</td></tr>
<tr><td>A3I5 Translucent Texture</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="3">ALPHA</td><td colspan="5">INDEX</td></tr>
<tr><td>Direct Color Texture</td><td>ALPHA</td><td colspan="5">BLUE</td><td colspan="5">GREEN</td><td colspan="5">RED</td></tr>
</table>

Compressed Texture Data (Note: 32-bit notation)

<table>
<tr><th>Format \ d</th><th>31</th><th>30</th><th>29</th><th>28</th><th>27</th><th>26</th><th>25</th><th>24</th><th>23</th><th>22</th><th>21</th><th>20</th><th>19</th><th>18</th><th>17</th><th>16</th><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>Compressed texture image</td><td colspan="2">T33</td><td colspan="2">T32</td><td colspan="2">T31</td><td colspan="2">T30</td><td colspan="2">T23</td><td colspan="2">T22</td><td colspan="2">T21</td><td colspan="2">T20</td><td colspan="2">T13</td><td colspan="2">T12</td><td colspan="2">T11</td><td colspan="2">T10</td><td colspan="2">T03</td><td colspan="2">T02</td><td colspan="2">T01</td><td colspan="2">T00</td></tr>
<tr><td>Compressed texture index</td><td colspan="16"></td><td>3/4</td><td>T</td><td colspan="14">Palette Address</td></tr>
</table>

Texture Palette Data

Format \ d	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Texture palette color	-	BLUE					GREEN					RED				

OAM Data

Format \ d	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OBJ attribute 0	Form		Color mode	Mosaic	OBJ mode		Double size	Rotation / Scaling	y-Coordinate							
OBJ attribute 1	Size		V-flip	H-flip				x-Coordinate								
			Affine transformation parameter number													
OBJ attribute 2	Palette Number				Order of Priority		Character name									
Affine transformation parameter PA	Sign	Integer part							Decimal part							
	Distance in x direction for same line															
Affine transformation parameter PB	Sign	Integer part							Decimal part							
	Distance in in x direction for next line															
Affine transformation parameter PC	Sign	Integer part							Decimal part							
	Distance in in y direction for same line															
Affine transformation parameter PD	Sign	Integer part							Decimal part							
	Distance in in y direction for next line															

Sound Data

Format \ d	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PCM 8 bit									Data 0							
PCM16 bit	Data 0															
ADPCM													Data 0			